

به نام خدا

**تمرین سری دوم برنامه نویسی پیشرفته**

دانشجو:

سیده فاطمه شاهرخی

شماره دانشجویی:

1402419

استاد مربوطه:

دکتر رشنو

آذر 1402

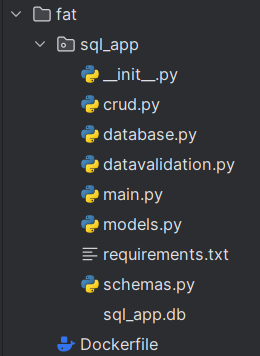
میخواهیم با استفاده از fastapi و دیتابیس sqlite سه جدول دانشجو، استاد، و درس درست کنیم.

برای این کار ابتدا پوشه ای به نام fat درست میکنیم که در این پوشه Dockerfile و پکیج sql\_app را ایجاد میکنیم

برای ساخت پکیج ابتدا فایلی به نام sql\_app ایجاد کرده و سپس درون آن یک فایل پایتون خالی به نام \_\_init\_\_.py قرار میدهیم تا تبدیل به پکیج شود سپس در پکیج 6 فایل زیر را ایجاد میکنیم:

***1. database.py, 2. models.py, 3. schemas.py, 4. crud.py,***

***5. datavalidation.py, 6. main.py***



فایل requirements.txt را برای داکر فایل و با استفاده از کتابخانه pipreqs ایجاد کردیم. این فایل آخرین ورژن نیازمندی هایمان برای پروژه (آخرین ورژن های کتابخانه هایی که در پروژه استفاده شدند) رو درون خودش دارد

دستور نصب: pip install pipreqs

سپس بعد از انجام دادن پروژه و زدن کد ها با استفاده از دستور cd fat/sql\_app به مسیر پکیجی که ساختیم میرویم و سپس دستور pipreqs . را وارد میکنیم سپس فایل requirements.txt برایمان ایجاد میشود

فایل sql\_app.db هم مربوط به دیتابیس و جدول ها هستش که با نصب اکستنشن میتوانیم در این فایل جدول ها را ببینیم

ما به SQLAlchemy نیاز داریم پس با دستورpip install SQLAlchemy آن را دانلود و نصب میکنیم .

در ادامه به توضیح فایل های پکیج و کد های آنها میپردازیم:

1. ***database.py***

در این فایل از SQLAlchemy برای ارتباط با پایگاه داده استفاده می‌کند. دیتابیس در اینجا می‌تواند SQLite یا PostgreSQL باشد، که از طریق متغیر SQLALCHEMY\_DATABASE\_URL مشخص می‌شود. و ما از SQLite استفاده میکنیم.

در این فایل ابتدا توابع و نیازمندی های دیتابیس رو import میکنیم:

# ایمپورت توابع مورد نیاز از SQLAlchemy  
from sqlalchemy import create\_engine # برای ایجاد موتور اتصال به دیتابیس  
from sqlalchemy.ext.declarative import declarative\_base # برای تعریف پایه برای مدل‌های دیتابیس  
from sqlalchemy.orm import sessionmaker # برای ایجاد کننده نشست برای ارتباط با دیتابیس

اولی برای ایجاد موتور اتصال کننده به پایگاه داد. در واقع به عنوان اولین مرحله در ایجاد ارتباط با پایگاه داده استفاده می‌شود.

دومی برای تعریف پایه تمام مدل های دیتابیس. declarative\_base برای تعریف یک کلاس پایه که تمام مدل‌ها از آن ارث بری میکنند، استفاده میشود. این کلاس پایه معمولاً شامل تنظیماتی مانند نام جدول دیتابیس و ویژگی های زمانی مانند تاریخ ایجاد و به روزرسانی است.

سومی هم برای ایجاد نشست (Session)هایی که برای ارتباط با پایگاه داده استفاده می‌شوند، استفاده می‌شود. در واقع نشست ها نقشی مانند یک بستر ایفا می‌کنند که به برنامه اجازه می‌دهند با دیتابیس ارتباط برقرار کنند، عملیات‌های CRUD را اجرا کنند و مثلا تغییرات را commit کنند. پس نشست (Session) به معنای یک جلسه ارتباط با پایگاه داده است که به کمک آن می توان عملیات های مختلفی را اجرا کرد.

این سه ماژول به طور کلی برای ایجاد و مدیریت اتصالات به دیتابیس در SQLAlchemy بکار میروند

# آدرس دیتابیس SQLite یا PostgreSQL  
SQLALCHEMY\_DATABASE\_URL = "sqlite:///./sql\_app.db"  
# SQLALCHEMY\_DATABASE\_URL = "postgresql://user:password@postgresserver/db"

در اینجا SQLALCHEMY\_DATABASE\_URL آدرس دیتابیس را مشخص میکند. اگر از SQLite استفاده شود، مسیر به ./sql\_app.db که یک فایل SQLite در همان دایرکتوری اجرایی است، و اگر از PostgreSQL استفاده شود، متغیر باید با آدرس متناسب با PostgreSQL تنظیم شود. و چون ما از دومی استفاده نمیکنیم میتوانیم آن را کامنت یا حذف کنیم

# ایجاد موتور اتصال به دیتابیس  
engine = create\_engine(  
 SQLALCHEMY\_DATABASE\_URL, connect\_args={"check\_same\_thread": False} # ارتباط در حالت تک نخ (برای SQLite)  
)

در اینجا create\_engine از SQLAlchemy برای ایجاد یک موتور اتصال به پایگاه داده استفاده می‌شود.

SQLALCHEMY\_DATABASE\_URL آدرس دیتابیس را مشخص میکند که بالاتر آن را تعیین کردیم

connect\_args={"check\_same\_thread": False} پارامتری است که برای SQLite تنظیم می‌شود تا ارتباط در حالت Thread ایجاد شود، به دلیل اینکه SQLite بهینه‌سازی نشده برای اتصالات Multi-thread است.

**:**

Thread:

یک Thread در زبان برنامه نویسی به عنوان یک فرآیند کوچکتر در داخل یک برنامه تعریف می‌شود.

Threadها برای اجرای همزمان (concurrent) عملیات‌ های مختلف در یک برنامه استفاده میشوند.

یک برنامه میتواند شامل یک یا چند Thread باشد که هر کدام میتوانند به صورت مستقل اجرا شوند.

Multi-thread:

وقتی که یک برنامه قادر به اجرای همزمان عملیات های مختلف در چند thread است، به ابن وضعیت Multi-thread میگویند

در این حالت، هر thread میتواند بخشی از کد برنامه را به صورت همزمان اجرا کند، بدون اینکه نیاز به انتظار برای پایان یک thread دیگر وجود داشته باشد

برای SQLite، وضعیت ارتباط در Multi-thread به خوبی پشتیبانی نمی‌شود و این به معنای آن است که اگرچه میتوانیم به SQLite از Multi-thread دسترسی داشته باشیم، اما انجام همزمان انها به درستی مدیریت نمیشود و ممکن است با مشکلاتی مثل ارور یا داده های پایگاه داده تخریب شده مواجه بشیم. برای همین، در اتصال به SQLite ، معمولاً از پارامترconnect\_args={"check\_same\_thread": False} استفاده میکنیم تا اجازه دهد که تمامی thread ها از یک اتصال اصلی و مشترک استفاده کنن و به این ترتیب مشکلات همگام سازی thread ها را برطرف کند

# تولید کننده نشست برای ارتباط با دیتابیس  
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

استفاده ازsessionmaker از SQLAlchemy به ما این امکان را می دهد که یک نشست (Session) استاندارد برای ارتباط با پایگاه داده ایجاد کنیم. در SQLAlchemy، Session به عنوان یک واسط برای انجام عملیات CRUD با پایگاه داده استفاده میشود. (عملیات CRUD شامل : Create, Read, Update, Delete)

با استفاده از sessionmaker، میتوانیم تنظیمات مختلفی را برای نشستمان تعیین کنیم، مانند خودکارکردن commit، autoflush، اتصال به engine و ...

به عبارت دیگر، sessionmaker یک کلاس است که به ما اجازه میدهد نشست های SQLAlchemy را با تنظیمات خاصی که میخواهیم ایجاد کنیم که بر اساس آن ها می‌توانیم با پایگاه داده تعامل داشته باشیم

**autocommit=False:** با استفاده از این،SQLAlchemy فقط تغییراتی که دستور commit دارند را به پایگاه داده ارسال میکند. با اینکار کنترل بیشتری بر انتقال داده ها به پایگاه داده داریم و از ایجاد تغییرات غیرمنتظره جلوگیری میشود.

**autoflush=False**: وقتی که این تنظیم غیرفعال است، SQLAlchemy منتظر تایید ما یرای دستور commit می ماند تا تغییرات را به پایگاه داده انتقال دهد. این به ما کنترل بیشتری بر روی زمانی که داده ها به پایگاه داده ارسال میشوند میدهد.

**bind=engine**: این پارامتر نشان میدهد که یک session که توسطsessionmaker ایجاد میشود به کدام موتور اتصال دیتابیس متصل میشود. موتور اتصال یا همان **engine** مسئول برقراری اتصال با پایگاه داده است و تمام عملیات ارسال و دریافت داده‌ها به کمک این موتور انجام می‌شود.

# تعریف پایه برای ایجاد مدل‌های دیتابیس  
Base = declarative\_base()

این قسمت کد‌ در SQLAlchemy به ما کمک میکند تا یک پایه اولیه برای تعریف مدل های دیتابیس ایجاد کنیم. این کار امکان ایجاد مدل های دیتابیسی که به راحتی م ‌توانند با پایگاه داده ارتباط برقرار کنند را فراهم می‌کند.

کل کد:

******

# ایمپورت توابع مورد نیاز از SQLAlchemy  
from sqlalchemy import create\_engine # برای ایجاد موتور اتصال به دیتابیس  
from sqlalchemy.ext.declarative import declarative\_base # برای تعریف پایه برای مدل‌های دیتابیس  
from sqlalchemy.orm import sessionmaker # برای ایجاد کننده نشست برای ارتباط با دیتابیس  
  
  
# آدرس دیتابیس SQLite یا PostgreSQL  
SQLALCHEMY\_DATABASE\_URL = "sqlite:///./sql\_app.db"  
# SQLALCHEMY\_DATABASE\_URL = "postgresql://user:password@postgresserver/db"  
  
# ایجاد موتور اتصال به دیتابیس  
engine = create\_engine(  
 SQLALCHEMY\_DATABASE\_URL, connect\_args={"check\_same\_thread": False} # ارتباط در حالت تک نخ (برای SQLite)  
)  
  
# تولید کننده نشست برای ارتباط با دیتابیس  
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)  
  
# تعریف پایه برای ایجاد مدل‌های دیتابیس  
Base = declarative\_base()  
  
# SessionLocal برای ایجاد نشست های جدید برای ارتباط با دیتابیس استفاده می‌شود.  
# autocommit=False به معنی غیرفعال بودن خودکار commit تراکنش هااست، که بهینه سازی عملیات دیتابیس را فراهم می‌کند.  
# autoflush=False به معنی غیرفعال بودن خودکار flush تراکنش هااست، که کنترل دقیق‌تری روی زمان ارسال داده ها به دیتابیس فراهم میکند.  
# bind=engine به معنی متصل کردن نشست‌های جدید به موتور دیتابیسی است که ارتباط را برقرار می‌کند.  
  
# Base یک کلاس پایه است که توسط declarative\_base() ایجاد شده است و مدل‌های دیتابیسی را تعریف و ایجاد می‌کند.  
# این کلاس برای تعریف جداول دیتابیس و نگهداری ساختارهای مرتبط با دیتابیس به کار می‌رود

1. ***models.py:***

این فایل برای تعریف مدل های داده هایی است که برای پایگاه داده مورد استفاده قرار می‌گیرند. در اینجا، مدل ها به شکل کلاس هایی با استفاده از SQLAlchemy تعریف شدند که ویژگی های مختلف داده ها را (مانند نام، نوع داده، و ...) تعیین میکنند و آنها را در جدول دیتابیس قرار میدهند. در واقع این فایل جدول دیتا بیس را میسازد

Column یه کلاس است که برای تعریف ویژگی های فیلد یا ستون در جدول پایگاه داده استفاده می شود. این ویژگی ها شامل نام ستون، نوع داده، کلید اصلی(primary\_key) و سایر تنظیمات مربوط به این ستون در جدول می باشند.

**کلید اصلی (**primary\_key**)**: اگر این پارامتر بر روی True تنظیم شود، این ستون به عنوان کلید اصلی جدول تعیین می‌شود که بیانگر یکتایی و اهمیت این ستون است.

پارامترindex=True در SQLAlchemy برای تعیین این استفاده می شود که آیا برای این ستون در پایگاه داده یک ایندکس باید ایجاد شود یا خیر. این ایندکس ها برای بهبود عملکرد عملیات جستجو، مرتب سازی و فیلتر کردن در پایگاه داده استفاده می‌شوند. عملکرد ایندکس در پایگاه داده: سرعت عملیات جستجو، ترتیب داده ها و ...

اول نیازمندی های مورد نیازمان را import میکنیم:

# ایمپورت کامپوننت های مورد نیاز از SQLAlchemy  
from sqlalchemy import Boolean, Column, ForeignKey, Integer, String  
# ایمپورت Base از ماژول database  
# Base از SQLAlchemy برای تعریف پایه‌ای که برای ساخت مدل‌های داده‌ای و جداول در پایگاه داده استفاده می‌شود، وارد می‌شود.  
from . database import Base  
from sqlalchemy.orm import relationship

جدول Student:

# STUDENT  
class Student(Base):  
 *"""  
 جدول مربوط به اطلاعات دانشجویان  
 """* \_\_tablename\_\_ = "student"  
 stid = Column(String, primary\_key=True) # شماره دانشجویی  
 fname = Column(String) # نام  
 lname = Column(String) # نام خانوادگی  
 father = Column(String) # نام پدر  
 birth = Column(String) # تاریخ تولد  
 ids = Column(String) # سریال شناسنامه  
 born\_city = Column(String) # شهر محل تولد مرکز استان  
 address = Column(String) # آدرس  
 postalcode = Column(Integer) # کد پستی  
 cphone = Column(String) # تلفن همراه  
 hphone = Column(String) # تلفن ثابت  
 department = Column(String) # دانشکده  
 major = Column(String) # رشته تحصیلی  
 married = Column(String) # وضعیت تاهل  
 id = Column(Integer) # کد ملی  
 scourse\_ids = Column(Integer) # کد دروس اخذ شده  
 lids = Column(Integer) # کد اساتید

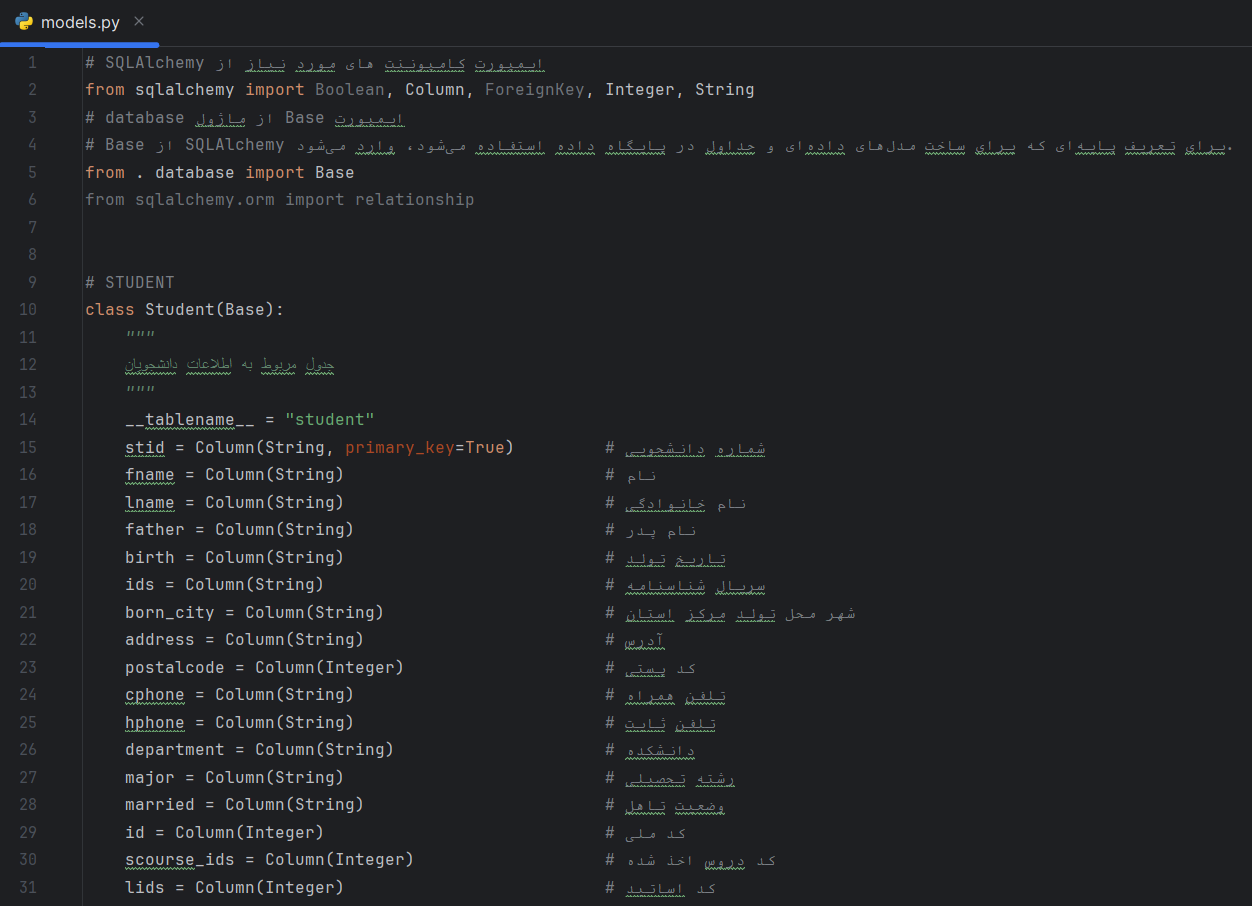
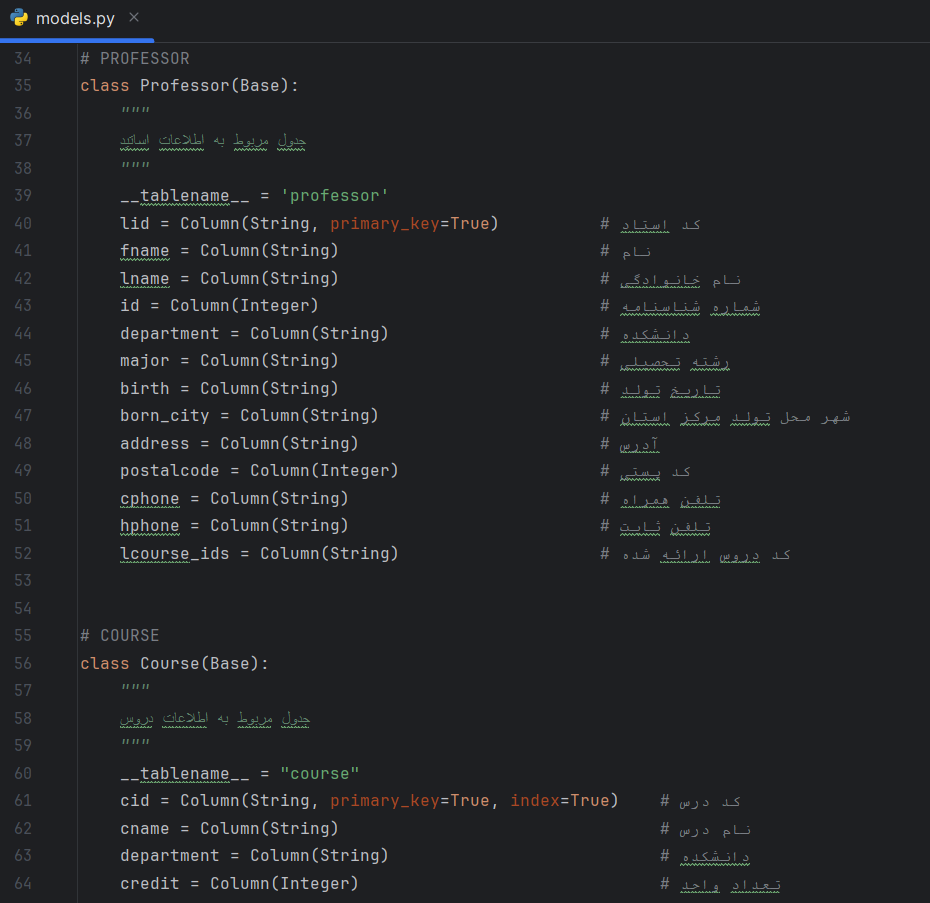
جدول Professor:

# PROFESSOR  
class Professor(Base):  
 *"""  
 جدول مربوط به اطلاعات اساتید  
 """* \_\_tablename\_\_ = 'professor'  
 lid = Column(String, primary\_key=True) # کد استاد  
 fname = Column(String) # نام  
 lname = Column(String) # نام خانوادگی  
 id = Column(Integer) # شماره شناسنامه  
 department = Column(String) # دانشکده  
 major = Column(String) # رشته تحصیلی  
 birth = Column(String) # تاریخ تولد  
 born\_city = Column(String) # شهر محل تولد مرکز استان  
 address = Column(String) # آدرس  
 postalcode = Column(Integer) # کد پستی  
 cphone = Column(String) # تلفن همراه  
 hphone = Column(String) # تلفن ثابت  
 lcourse\_ids = Column(String) # کد دروس ارائه شده

جدول Course:

# COURSE  
class Course(Base):  
 *"""  
 جدول مربوط به اطلاعات دروس  
 """* \_\_tablename\_\_ = "course"  
 cid = Column(String, primary\_key=True, index=True) # کد درس  
 cname = Column(String) # نام درس  
 department = Column(String) # دانشکده  
 credit = Column(Integer) # تعداد واحد

کل کد:



# ایمپورت کامپوننت های مورد نیاز از SQLAlchemy  
from sqlalchemy import Boolean, Column, ForeignKey, Integer, String  
# ایمپورت Base از ماژول database  
# Base از SQLAlchemy برای تعریف پایه‌ای که برای ساخت مدل‌های داده‌ای و جداول در پایگاه داده استفاده می‌شود، وارد می‌شود.  
from . database import Base  
from sqlalchemy.orm import relationship  
  
  
# STUDENT  
class Student(Base):  
 *"""  
 جدول مربوط به اطلاعات دانشجویان  
 """* \_\_tablename\_\_ = "student"  
 stid = Column(String, primary\_key=True) # شماره دانشجویی  
 fname = Column(String) # نام  
 lname = Column(String) # نام خانوادگی  
 father = Column(String) # نام پدر  
 birth = Column(String) # تاریخ تولد  
 ids = Column(String) # سریال شناسنامه  
 born\_city = Column(String) # شهر محل تولد مرکز استان  
 address = Column(String) # آدرس  
 postalcode = Column(Integer) # کد پستی  
 cphone = Column(String) # تلفن همراه  
 hphone = Column(String) # تلفن ثابت  
 department = Column(String) # دانشکده  
 major = Column(String) # رشته تحصیلی  
 married = Column(String) # وضعیت تاهل  
 id = Column(Integer) # کد ملی  
 scourse\_ids = Column(Integer) # کد دروس اخذ شده  
 lids = Column(Integer) # کد اساتید  
  
  
# PROFESSOR  
class Professor(Base):  
 *"""  
 جدول مربوط به اطلاعات اساتید  
 """* \_\_tablename\_\_ = 'professor'  
 lid = Column(String, primary\_key=True) # کد استاد  
 fname = Column(String) # نام  
 lname = Column(String) # نام خانوادگی  
 id = Column(Integer) # شماره شناسنامه  
 department = Column(String) # دانشکده  
 major = Column(String) # رشته تحصیلی  
 birth = Column(String) # تاریخ تولد  
 born\_city = Column(String) # شهر محل تولد مرکز استان  
 address = Column(String) # آدرس  
 postalcode = Column(Integer) # کد پستی  
 cphone = Column(String) # تلفن همراه  
 hphone = Column(String) # تلفن ثابت  
 lcourse\_ids = Column(String) # کد دروس ارائه شده

# COURSE  
class Course(Base):  
 *"""  
 جدول مربوط به اطلاعات دروس  
 """* \_\_tablename\_\_ = "course"  
 cid = Column(String, primary\_key=True, index=True) # کد درس  
 cname = Column(String) # نام درس  
 department = Column(String) # دانشکده  
 credit = Column(Integer) # تعداد واحد

1. ***schemas.py***

این فایل برای تعریف ساختار جداول و مدل‌های داده ای استفاده میشود و شامل تعریف کلاس های Student،Professor و Course با استفاده از کتابخانه pydantic است که هر کدام شامل فیلدها و اطلاعات مرتبط با خودشان هستند. همچنین با کمک این فایل فرمت داده ها چک میشوند و از درست بودنشان مطمئن میشویم.

در ابتدا نیازمندیمان را import میکنیم. با وارد کردنBaseModel از pydantic میتوانیم سه کلاس استاد،ردانشجو و درس را تعریف کرده و از ویژگی های اعتبارسنجی و تبدیل داده ها که توسطpydantic ارائه می‌شود استفاده کنیم تا داده ها را مدیریت کرده و به صورت به همدیگر مرتبط کنیم.

from pydantic import BaseModel

چون از ما خواسته شده در پاسخ فقط 4 فیلد اول نمایش داده شوند پس برای دانشجو و استاد اول یک کلاس درست میکنیم که حاوی آن 4 فیلد هستند سپس یک کلاس دیگر درست میکنیم که از این کلاس ارث بری میکند و شامل همه فیلد ها هستند

کلاس دانشجو:

# STUDENT

class Student\_set(BaseModel):  
 stid: str # شماره دانشجویی  
 fname: str # نام  
 lname: str # نام خانوادگی  
 father: str # نام پدر  
  
  
class Student(Student\_set):  
 stid: str # شماره دانشجویی  
 fname: str # نام  
 lname: str # نام خانوادگی  
 father: str # نام پدر  
 birth: str # تاریخ تولد  
 ids: str # سریال شناسنامه  
 born\_city: str # شهر محل تولد مرکز استان  
 address: str # آدرس  
 postalcode: int # کد پستی  
 cphone: str # تلفن همراه  
 hphone: str # تلفن ثابت  
 department: str # دانشکده  
 major: str # رشته تحصیلی  
 married: str # وضعیت تاهل  
 id: int # کد ملی  
 scourse\_ids: int # کد دروس اخذ شده  
 lids: int # کد اساتید

کلاس استاد:

# PROFESSOR  
class Professor\_set(BaseModel):  
 lid: str # کد استاد  
 fname: str # نام  
 lname: str # نام خانوادگی  
 id: int # کد ملی  
  
class Professor(Professor\_set):  
 lid: str # کد استاد  
 fname: str # نام  
 lname: str # نام خانوادگی  
 id: int # کد ملی  
 department: str # دانشکده  
 major: str # رشته تحصیلی  
 birth: str # تاریخ تولد  
 born\_city: str # شهر محل تولد مرکز استان  
 address: str # آدرس  
 postalcode: int # کد پستی  
 cphone: str # تلفن همراه  
 hphone: str # تلفن ثابت  
 lcourse\_ids: str # کد دروس ارئه شده

کلاس درس:

# COURSE  
class Course(BaseModel):  
 cid: str # کد درس  
 cname: str # نام درس  
 department: str # دانشکده  
 credit: int # تعداد واحد

کلاس درس چون فقط 4 فیلد دارد فقط یک کلاس برایش ایجاد میکنیم

کل کد:

from pydantic import BaseModel  
  
  
# STUDENT  
class Student\_set(BaseModel):  
 stid: str # شماره دانشجویی  
 fname: str # نام  
 lname: str # نام خانوادگی  
 father: str # نام پدر  
  
  
class Student(Student\_set):  
 stid: str # شماره دانشجویی  
 fname: str # نام  
 lname: str # نام خانوادگی  
 father: str # نام پدر  
 birth: str # تاریخ تولد  
 ids: str # سریال شناسنامه  
 born\_city: str # شهر محل تولد مرکز استان  
 address: str # آدرس  
 postalcode: int # کد پستی  
 cphone: str # تلفن همراه  
 hphone: str # تلفن ثابت  
 department: str # دانشکده  
 major: str # رشته تحصیلی  
 married: str # وضعیت تاهل  
 id: int # کد ملی  
 scourse\_ids: int # کد دروس اخذ شده  
 lids: int # کد اساتید  
  
  
  
  
# PROFESSOR  
class Professor\_set(BaseModel):  
 lid: str # کد استاد  
 fname: str # نام  
 lname: str # نام خانوادگی  
 id: int # کد ملی  
  
  
class Professor(Professor\_set):  
 lid: str # کد استاد  
 fname: str # نام  
 lname: str # نام خانوادگی  
 id: int # کد ملی  
 department: str # دانشکده  
 major: str # رشته تحصیلی  
 birth: str # تاریخ تولد  
 born\_city: str # شهر محل تولد مرکز استان  
 address: str # آدرس  
 postalcode: int # کد پستی  
 cphone: str # تلفن همراه  
 hphone: str # تلفن ثابت  
 lcourse\_ids: str # کد دروس ارئه شده  
  
  
  
  
# COURSE  
class Course(BaseModel):  
 cid: str # کد درس  
 cname: str # نام درس  
 department: str # دانشکده  
 credit: int # تعداد واحد

در واقع این فایل نوع داده ورودی هر ستون از جدول را مشخص میکند و برای محدود کردن نمایش فیلد ها در خروجی کمک میکند

1. ***crud.py:***

فایل CRUD در برنامه نویسی به یک فایل یا بخش از کد گفته می‌شود که شامل توابع و عملیات برای ایجاد (Create)، خواندن (Read)، به روزرسانی (Update) و حذف (Delete) داده ها در یک پایگاه داده است. این فایل ها برای مدیریت داده ها و ارتباط با پایگاه داده در برنامه ها استفاده می شوند و اصطلاحا به آن CRUD گفته میشود که این عبارات مخفف مراحل اصلی مدیریت داده ها هستند.

در واقع فایل crud ما برای ایجاد ارتباط با پایگاه داده و تعریف عملیات مورد نیاز برای مدیریت دانشجویان، استادان و دروس است

ابتدا نیازمندی هایمان را import میکنیم:

# ایجاد ارتباط با پایگاه داده و تعریف عملیات مورد نیاز برای مدیریت دانشجویان، استادان و دروس  
  
# وارد کردن کلاس Session از SQLAlchemy ORM برای مدیریت نشست‌های پایگاه داده  
from sqlalchemy.orm import Session  
from . import models, schemas # وارد کردن مدل‌ها و اسکیماها از ماژول فعلی  
from fastapi import HTTPException # وارد کردن استثنائات HTTP از FastAPI

سپس برای هر کدام از جداول دانشجو، استاد و درس چهار تابع create, get, update(put), delete را مینویسیم

**دانشجو:**

تابع create:

این تابع برای ایجاد یک دانشجو جدید در پایگاه داده استفاده می‌شود.

در ورودی، نیاز به یک نشست دیتابیس (db) داریم که به عنوان پارامتر ارسال می‌شود، و همچنین یک شی student از نوعschemas.Student که شامل اطلاعات دانشجو برای ثبت می‌باشد.

def create\_student(db: Session, student: schemas.Student):

سپس یک شی به نام db\_student از مدل Student تعریف میکنیم و اطلاعات دانشجو از شی Student به آن منتقل میشود:

db\_student = models.Student(  
 stid=student.stid,  
 fname=student.fname,  
 lname=student.lname,  
 father=student.father,  
 birth=student.birth,  
 ids=student.ids,  
 born\_city=student.born\_city,  
 address=student.address,  
 postalcode=student.postalcode,  
 cphone=student.cphone,  
 hphone=student.hphone,  
 department=student.department,  
 major=student.major,  
 married=student.married,  
 id=student.id,  
 scourse\_ids=student.scourse\_ids,  
 lids=student.lids  
)

سپس شی db\_student که اکنون حاوی اطلاعات دانشجو است را به نشست دیتابیس (db) اضافه میکنیم تا برای ثبت در پایگاه داده آماده شود

db.add(db\_student) # افزودن دانشجو به نشست

سپس با فراخوانی db.commit() تمامی تغییرات انجام شده یعنی همان افزودن دانشجو به پایگاه داده ثبت و اعمال میشود

db.commit() # ثبت تراکنش در پایگاه داده

پس از ثبت، شی db\_student به روزرسانی میشود تا اطلاعات به روزشده را دریافت کند

db.refresh(db\_student) # به‌روزرسانی شی دانشجو برای دریافت مقادیر به‌روز شده

در نهایت شی دانشجوی ایجاد شده با تمامی اطلاعات و به روزرسانی های لازم به عنوان خروجی تابع باز می‌گردد

return db\_student # بازگشت شی دانشجو ایجاد شده

این تابع از ماژول schemas برای تعریف داده ها و از models برای تعامل با پایگاه داده استفاده میکند و با استفاده از SQLAlchemy orm، عملیات ایجاد دانشجو، ثبت آن در پایگاه داده، به‌روزرسانی اطلاعات و بازگشت شی دانشجوی ایجاد شده را انجام میدهد

کل کد:

def create\_student(db: Session, student: schemas.Student):  
 *"""  
 ایجاد یک دانشجو جدید در پایگاه داده.  
  
 :param db: نشست دیتابیس  
 :param student: اطلاعات دانشجو برای ثبت  
 :return: شی دانشجوی ایجاد شده  
 """* db\_student = models.Student(  
 stid=student.stid,  
 fname=student.fname,  
 lname=student.lname,  
 father=student.father,  
 birth=student.birth,  
 ids=student.ids,  
 born\_city=student.born\_city,  
 address=student.address,  
 postalcode=student.postalcode,  
 cphone=student.cphone,  
 hphone=student.hphone,  
 department=student.department,  
 major=student.major,  
 married=student.married,  
 id=student.id,  
 scourse\_ids=student.scourse\_ids,  
 lids=student.lids  
 )  
 db.add(db\_student) # افزودن دانشجو به نشست  
 db.commit() # ثبت تراکنش در پایگاه داده  
 db.refresh(db\_student) # به‌روزرسانی شی دانشجو برای دریافت مقادیر به‌روز شده  
 return db\_student # بازگشت شی دانشجو ایجاد شده

تابع get:

تابع get\_student برای دریافت اطلاعات دانشجو از پایگاه داده بر اساس شماره دانشجویی است

ورودی های آن شامل یک نشست دیتابیس (db) و یک رشته (student\_stid) است که شماره دانشجویی مورد نظر را مشخص میکند‌ و از Session وmodels.Student\_set برای تعریف داده ها و تعامل با پایگاه داده استفاده می‌کند

schemas.Student\_set شامل چهار فیلد اول از اطلاعات دانشجو است (شماره دانشجویی، نام، نام خانوادگی، نام پدر)

def get\_student(db: Session, student\_stid: str) -> schemas.Student\_set:

با استفاده ازdb.query(models.Student) یک query برای جستجوی اطلاعات دانشجو از جدول مدل Student تعریف میکنیم

استفاده ازdb.query(models.Student) در SQLAlchemy به معنای تعریف یک query SQL برای جستجوی اطلاعات دانشجو از جدول مدلStudentاست. این روش از ORM امکان استفاده از شیStudentرا برای عملیات مختلفی مانند جستجو، اضافه کردن، به روزرسانی و حذف داده ها فراهم می کند

سپس با استفاده از .filter(models.Student.stid == student\_stid) اطلاعات آن دانشجویی که شماره دانشجوییش با مقدارstudent\_stid مساوی است، فیلتر میکنیم و با استفاده از .first() هم اگر چند دانشجو با شرایط فیلتر شده مطابقت داشته باشند اولین دانشجو به عنوان خروجی بازگردانده میشود و در آخر پاسخ را return میکنیم

return db.query(models.Student).filter(models.Student.stid == student\_stid).first()

کل کد:

def get\_student(db: Session, student\_stid: str) -> schemas.Student\_set:  
 *"""  
 دریافت یک دانشجو براساس شماره دانشجویی و بازگرداندن فقط چهار فیلد اول  
  
 :param db: نشست دیتابیس  
 :param student\_stid: شماره دانشجویی  
 :return: شی دانشجوی با چهار فیلد اول  
 """* db\_student = db.query(models.Student).filter(models.Student.stid == student\_stid).first()  
 if db\_student:  
 student\_set = schemas.Student\_set(  
 stid=db\_student.stid,  
 fname=db\_student.fname,  
 lname=db\_student.lname,  
 father=db\_student.father  
 )  
 return student\_set  
 else:  
 raise HTTPException(status\_code=404, detail="stid not found")

تابع update:

دو پارامتر اول این تابع یعنی db و student\_stid ورودی و پارامتر سوم یعنی new\_student\_data خروجی است که یک شی از نوع schemas.Student\_set است که شامل چهار فیلد اول می باشد

def update\_student(db: Session, student\_stid: str, new\_student\_data: schemas.Student\_set):

در خط

db.query(models.Student).filter(models.Student.stid == student\_stid).first():

یک query به دیتابیس زده میشود برای جستجوی دانشجو براساس شماره دانشجویی (student\_stid) . این query به وسیله ORM SQLAlchemy ساخته میشود تا اطلاعات متناظر با شماره دانشجویی داده شده را از جدول Student بخواند

db\_student = db.query(models.Student).filter(models.Student.stid == student\_stid).first()

سپس بررسی میشود که آیا دانشجو با شماره دانشجویی داده شده وجود دارد یا خیر

در صورت وجود دانشجو، حلقه

for attr, value in vars(new\_student\_data).items():

ویژگی های جدید دانشجو را چک میکند و هر ویژگی را با مقدار متناظر آن از ویژگی های جدید آپدیت میکند

vars(new\_student\_data) از تابع vars() استفاده میکند تا ویژگی‌های متغیرnew\_student\_data (که یک شیء از نوع schemas.Student است) را به صورت یک دیکشنری بازگرداند.

دستور setattr(db\_student, attr, value) برای هر ویژگی (attr) و مقدار (value) استخراج شده از دیکشنری، ویژگی متناظر شی db\_student را با مقدار جدید تنظیم می‌کند. به عبارت دیگر، این دستور ویژگی‌های شی db\_student را با اطلاعات جدید ازnew\_student\_data به‌روز می‌کند.

سپس بوسیه db.commit() تراکنش در پایگاه داده ثبت میشود تا تغییرات اعمال شوند و بعد با db.refresh(db\_student) شی دانشجو آپدیت میشود و اطلاعات جدید در پایگاه داده قرار میگیرند. درنهایت شی به روزرسانی شده return میشود

if db\_student:  
 # به‌روزرسانی هر ویژگی از دانشجو با مقادیر جدید  
 for attr, value in vars(new\_student\_data).items():  
 setattr(db\_student, attr, value)  
 db.commit() # ثبت تراکنش برای ذخیره تغییرات  
 db.refresh(db\_student) # به‌روزرسانی شی دانشجو برای دریافت مقادیر به‌روز شده  
 return db\_student # بازگشت شی دانشجوی به‌روزرسانی شده

اگر دانشجو با شماره دانشجویی داده شده یافت نشود یک استثناء HTTP با کد 404 و پیام "stid not found" پرتاب میشود تا به عنوان پاسخ به درخواست مورد نظر برگردانده شود

else:  
 raise HTTPException(status\_code=404, detail="stid not found")

کل کد:

def update\_student(db: Session, student\_stid: str, new\_student\_data: schemas.Student\_set):  
 *"""  
 به روزرسانی اطلاعات یک دانشجو براساس شماره دانشجویی  
  
 :param db: نشست دیتابیس  
 :param student\_stid: شماره دانشجویی  
 :param new\_student\_data: اطلاعات جدید دانشجو برای به روزرسانی  
 :return: شی دانشجوی به روزرسانی شده  
 """* db\_student = db.query(models.Student).filter(models.Student.stid == student\_stid).first()  
 if db\_student:  
 # به‌روزرسانی هر ویژگی از دانشجو با مقادیر جدید  
 for attr, value in vars(new\_student\_data).items():  
 setattr(db\_student, attr, value)  
 db.commit() # ثبت تراکنش برای ذخیره تغییرات  
 db.refresh(db\_student) # به‌روزرسانی شی دانشجو برای دریافت مقادیر به‌روز شده  
 return db\_student # بازگشت شی دانشجوی به‌روزرسانی شده  
 else:  
 raise HTTPException(status\_code=404, detail="stid not found")

تابع delete:

این تابع دو شی ورودی دارد. db که نشست با دیتابیس را برقرار میکند و student\_stid که با توجه به شماره دانشجویی قسمتی که باید حذف شود را نشان میدهد

def delete\_student(db: Session, student\_stid: str):

در خط

db\_student = db.query(models.Student).filter(models.Student.stid == student\_stid).first()

از SQLAlchemy ORM برای جست و جوی داتشجو با شماره دانشجویی وارد شده از جدول مدل Student استفاده میشود

db\_student = db.query(models.Student).filter(models.Student.stid == student\_stid).first()

سپس بررسی میشود که آیا دانشجو با شماره دانشجویی مورد نظر وجود دارد یا خیر. اگر وجود نداشته باشد مقدار None برگردانده میشود اگر وجود داشته باشد در خط بعد با استفاده از db.delete(db\_student) در صورتی که دانشجو با شماره دانشجویی موجود باشد، این دستور شی دانشجو را حذف میکند و پس از حذف شی دستور db.commit() تراکنش را در پایگاه داده ثبت میکند تا تغییرات اعمال شود

if db\_student:  
 db.delete(db\_student) # حذف شی دانشجو از نشست  
 db.commit() # ثبت تراکنش برای اعمال تغییرات در پایگاه داده  
 return db\_student  
return None # اگر دانشجو پیدا نشد بازگرداندن None

کل کد:

def delete\_student(db: Session, student\_stid: str):  
 *"""  
 حذف یک دانشجو براساس شماره دانشجویی  
  
 :param db: نشست دیتابیس  
 :param student\_stid: شماره دانشجویی  
 :return: پاسخ JSON موفقیت‌آمیز در صورت حذف موفقیت‌آمیز  
 """* db\_student = db.query(models.Student).filter(models.Student.stid == student\_stid).first()  
 if db\_student:  
 db.delete(db\_student) # حذف شی دانشجو از نشست  
 db.commit() # ثبت تراکنش برای اعمال تغییرات در پایگاه داده  
 return db\_student  
 return None # اگر دانشجو پیدا نشد بازگرداندن None

برای عملیات crud استاد و درس هم مانند همین ها عمل میکنیم

**استاد:**

**تابع create:**

def create\_professor(db: Session, professor: schemas.Professor):  
 *"""  
 ایجاد یک استاد جدید در پایگاه داده  
 :param db: نشست دیتابیس  
 :param professor: اطلاعات استاد برای ثبت  
 :return: شی استاد ایجاد شده  
 """* db\_professor = models.Professor(  
 lid=professor.lid,  
 fname=professor.fname,  
 lname=professor.lname,  
 id=professor.id,  
 department=professor.department,  
 major=professor.major,  
 birth=professor.birth,  
 born\_city=professor.born\_city,  
 address=professor.address,  
 postalcode=professor.postalcode,  
 cphone=professor.cphone,  
 hphone=professor.hphone,  
 lcourse\_ids=professor.lcourse\_ids,  
 )  
 db.add(db\_professor) # افزودن استاد به نشست  
 db.commit() # ثبت تراکنش در پایگاه داده  
 db.refresh(db\_professor) # به‌روزرسانی شی استاد برای دریافت مقادیر به‌روز شده  
 return db\_professor # بازگشت شی استاد ایجاد شده

**تابع get:**

def get\_professor(db: Session, professor\_lid: str) -> schemas.Professor\_set:  
 *"""  
 دریافت یک استاد براساس کد استاد  
 :param db: نشست دیتابیس  
 :param professor\_lid: کد استاد  
 :return: شی استاد مطابق با کد استاد داده شده  
 """* db\_professor = db.query(models.Professor).filter(models.Professor.lid == professor\_lid).first()  
 if db\_professor:  
 professor\_set = schemas.Professor\_set(  
 lid=db\_professor.lid,  
 fname=db\_professor.fname,  
 lname=db\_professor.lname,  
 id=db\_professor.id  
 )  
 return professor\_set  
 else:  
 raise HTTPException(status\_code=404, detail="lid not found")

**تابع update:**

def update\_professor(db: Session, professor\_lid: str, new\_professor\_data: schemas.Professor\_set):  
 *"""  
 به روزرسانی اطلاعات یک استاد براساس کد استاد  
  
 :param db: نشست دیتابیس  
 :param professor\_lid: کد استادی  
 :param new\_professor\_data: اطلاعات جدید استاد برای به روزرسانی  
 :return: شی استاد به روزرسانی شده  
 """* db\_professor = db.query(models.Professor).filter(models.Professor.lid == professor\_lid).first()  
 if db\_professor:  
 # به روزرسانی هر ویژگی از استاد با مقادیر جدید  
 for attr, value in vars(new\_professor\_data).items():  
 setattr(db\_professor, attr, value)  
 db.commit() # ثبت تراکنش برای ذخیره تغییرات  
 db.refresh(db\_professor) # به روزرسانی شی استاد برای دریافت مقادیر به‌روز شده  
 return db\_professor # بازگشت شی استاد به روزرسانی شده  
 raise HTTPException(status\_code=404, detail="lid not found")

**تابع delete:**

def delete\_professor(db: Session, professor\_lid: str):  
 *"""  
 حذف یک استاد براساس کد استاد  
  
 :param db: نشست دیتابیس  
 :param professor\_lid: کد استاد  
 :return: پاسخ JSON موفقیت آمیز در صورت حذف موفقیت آمیز  
 """* db\_professor = db.query(models.Professor).filter(models.Professor.lid == professor\_lid).first()  
 if db\_professor:  
 db.delete(db\_professor) # حذف شی استاد از نشست  
 db.commit() # ثبت تراکنش برای اعمال تغییرات در پایگاه داده  
 return db\_professor # بازگرداندن شی استاد حذف شده  
 return None # اگر استاد پیدا نشد بازگرداندن None

**درس:**

**تابع create:**

def create\_course(db: Session, course: schemas.Course):  
 *"""  
 ایجاد یک درس جدید در پایگاه داده  
  
 :param db: نشست دیتابیس  
 :param course: اطلاعات درس برای ثبت  
 :return: شی درس ایجاد شده  
 """* db\_course = models.Course(  
 cid=course.cid,  
 cname=course.cname,  
 department=course.department,  
 credit=course.credit  
 )  
 db.add(db\_course) # افزودن درس به نشست  
 db.commit() # ثبت تراکنش در پایگاه داده  
 db.refresh(db\_course) # به‌روزرسانی شی درس برای دریافت مقادیر به‌روز شده  
 return db\_course # بازگشت شی درس ایجاد شده

**تابع get:**

def get\_course(db: Session, course\_cid: str) -> models.Course:  
 *"""  
 دریافت یک درس براساس کد درسی  
  
 :param db: نشست دیتابیس  
 :param course\_cid: کد درسی  
 :return: شی درس مطابق با کد درسی داده شده  
 """* return db.query(models.Course).filter(models.Course.cid == course\_cid).first()

**تابع update:**

def update\_course(db: Session, course\_cid: str, new\_course\_data: schemas.Course):  
 *"""  
 به‌روزرسانی اطلاعات یک درس براساس کد درس  
  
 :param db: نشست دیتابیس  
 :param course\_cid: کد درسی  
 :param new\_course\_data: اطلاعات جدید درس برای به روزرسانی  
 :return: شی درس به روزرسانی شده  
 """* db\_course = db.query(models.Course).filter(models.Course.cid == course\_cid).first()  
 if db\_course:  
 # به روزرسانی هر ویژگی از درس با مقادیر جدید  
 for attr, value in vars(new\_course\_data).items():  
 setattr(db\_course, attr, value)  
 db.commit() # ثبت تراکنش برای ذخیره تغییرات  
 db.refresh(db\_course) # به روزرسانی شی درس برای دریافت مقادیر به روز شده  
 return db\_course # بازگشت شی درس به روزرسانی شده  
 raise HTTPException(status\_code=404, detail="cid not found")

**تابع delete:**

def delete\_course(db: Session, course\_cid: str):  
 *"""  
 حذف یک درس براساس کد درسی  
  
 :param db: نشست دیتابیس  
 :param course\_cid: کد درسی  
 :return: شی درس حذف شده  
 """* db\_course = db.query(models.Course).filter(models.Course.cid == course\_cid).first()  
 if db\_course:  
 db.delete(db\_course) # حذف شی درس از نشست  
 db.commit() # ثبت تراکنش برای اعمال تغییرات در پایگاه داده  
 return db\_course # بازگرداندن شی درس حذف شده  
 return None # اگر درس پیدا نشد بازگرداندن None

کل کد:

# ایجاد ارتباط با پایگاه داده و تعریف عملیات مورد نیاز برای مدیریت دانشجویان، استادان و دروس  
  
# وارد کردن کلاس Session از SQLAlchemy ORM برای مدیریت نشست‌های پایگاه داده  
from sqlalchemy.orm import Session  
from . import models, schemas # وارد کردن مدل‌ها و اسکیماها از ماژول فعلی  
from fastapi import HTTPException # وارد کردن استثنائات HTTP از FastAPI  
  
  
  
# STUDENT  
  
def create\_student(db: Session, student: schemas.Student):  
 *"""  
 ایجاد یک دانشجو جدید در پایگاه داده  
 :param db: نشست دیتابیس  
 :param student: اطلاعات دانشجو برای ثبت  
 :return: شی دانشجوی ایجاد شده  
 """* db\_student = models.Student(  
 stid=student.stid,  
 fname=student.fname,  
 lname=student.lname,  
 father=student.father,  
 birth=student.birth,  
 ids=student.ids,  
 born\_city=student.born\_city,  
 address=student.address,  
 postalcode=student.postalcode,  
 cphone=student.cphone,  
 hphone=student.hphone,  
 department=student.department,  
 major=student.major,  
 married=student.married,  
 id=student.id,  
 scourse\_ids=student.scourse\_ids,  
 lids=student.lids  
 )  
 db.add(db\_student) # افزودن دانشجو به نشست  
 db.commit() # ثبت تراکنش در پایگاه داده  
 db.refresh(db\_student) # به‌روزرسانی شی دانشجو برای دریافت مقادیر به‌روز شده  
 return db\_student # بازگشت شی دانشجو ایجاد شده  
  
  
  
def get\_student(db: Session, student\_stid: str) -> schemas.Student\_set:  
 *"""  
 دریافت یک دانشجو براساس شماره دانشجویی و بازگرداندن فقط چهار فیلد اول  
  
 :param db: نشست دیتابیس  
 :param student\_stid: شماره دانشجویی  
 :return: شی دانشجوی با چهار فیلد اول  
 """* db\_student = db.query(models.Student).filter(models.Student.stid == student\_stid).first()  
 if db\_student:  
 student\_set = schemas.Student\_set(  
 stid=db\_student.stid,  
 fname=db\_student.fname,  
 lname=db\_student.lname,  
 father=db\_student.father  
 )  
 return student\_set  
 else:  
 raise HTTPException(status\_code=404, detail="stid not found")  
  
  
  
  
  
def update\_student(db: Session, student\_stid: str, new\_student\_data: schemas.Student\_set):  
 *"""  
 به روزرسانی اطلاعات یک دانشجو براساس شماره دانشجویی  
  
 :param db: نشست دیتابیس  
 :param student\_stid: شماره دانشجویی  
 :param new\_student\_data: اطلاعات جدید دانشجو برای به روزرسانی  
 :return: شی دانشجوی به روزرسانی شده  
 """* db\_student = db.query(models.Student).filter(models.Student.stid == student\_stid).first()  
 if db\_student:  
 # به‌روزرسانی هر ویژگی از دانشجو با مقادیر جدید  
 for attr, value in vars(new\_student\_data).items():  
 setattr(db\_student, attr, value)  
 db.commit() # ثبت تراکنش برای ذخیره تغییرات  
 db.refresh(db\_student) # به‌روزرسانی شی دانشجو برای دریافت مقادیر به‌روز شده  
 return db\_student # بازگشت شی دانشجوی به‌روزرسانی شده  
 else:  
 raise HTTPException(status\_code=404, detail="stid not found")  
  
  
  
  
def delete\_student(db: Session, student\_stid: str):  
 *"""  
 حذف یک دانشجو براساس شماره دانشجویی  
  
 :param db: نشست دیتابیس  
 :param student\_stid: شماره دانشجویی  
 :return: پاسخ JSON موفقیت‌آمیز در صورت حذف موفقیت‌آمیز  
 """* db\_student = db.query(models.Student).filter(models.Student.stid == student\_stid).first()  
 if db\_student:  
 db.delete(db\_student) # حذف شی دانشجو از نشست  
 db.commit() # ثبت تراکنش برای اعمال تغییرات در پایگاه داده  
 return db\_student  
 return None # اگر دانشجو پیدا نشد بازگرداندن None  
  
  
# PROFESSOR  
  
def create\_professor(db: Session, professor: schemas.Professor):  
 *"""  
 ایجاد یک استاد جدید در پایگاه داده  
  
 :param db: نشست دیتابیس  
 :param professor: اطلاعات استاد برای ثبت  
 :return: شی استاد ایجاد شده  
 """* db\_professor = models.Professor(  
 lid=professor.lid,  
 fname=professor.fname,  
 lname=professor.lname,  
 id=professor.id,  
 department=professor.department,  
 major=professor.major,  
 birth=professor.birth,  
 born\_city=professor.born\_city,  
 address=professor.address,  
 postalcode=professor.postalcode,  
 cphone=professor.cphone,  
 hphone=professor.hphone,  
 lcourse\_ids=professor.lcourse\_ids,  
 )  
 db.add(db\_professor) # افزودن استاد به نشست  
 db.commit() # ثبت تراکنش در پایگاه داده  
 db.refresh(db\_professor) # به‌روزرسانی شی استاد برای دریافت مقادیر به‌روز شده  
 return db\_professor # بازگشت شی استاد ایجاد شده  
  
def get\_professor(db: Session, professor\_lid: str) -> schemas.Professor\_set:  
 *"""  
 دریافت یک استاد براساس کد استاد  
  
 :param db: نشست دیتابیس  
 :param professor\_lid: کد استاد  
 :return: شی استاد مطابق با کد استاد داده شده  
 """* db\_professor = db.query(models.Professor).filter(models.Professor.lid == professor\_lid).first()  
 if db\_professor:  
 professor\_set = schemas.Professor\_set(  
 lid=db\_professor.lid,  
 fname=db\_professor.fname,  
 lname=db\_professor.lname,  
 id=db\_professor.id  
 )  
 return professor\_set  
 else:  
 raise HTTPException(status\_code=404, detail="lid not found")  
  
  
  
  
  
def update\_professor(db: Session, professor\_lid: str, new\_professor\_data: schemas.Professor\_set):  
 *"""  
 به روزرسانی اطلاعات یک استاد براساس کد استاد  
  
 :param db: نشست دیتابیس  
 :param professor\_lid: کد استادی  
 :param new\_professor\_data: اطلاعات جدید استاد برای به روزرسانی  
 :return: شی استاد به روزرسانی شده  
 """* db\_professor = db.query(models.Professor).filter(models.Professor.lid == professor\_lid).first()  
 if db\_professor:  
 # به روزرسانی هر ویژگی از استاد با مقادیر جدید  
 for attr, value in vars(new\_professor\_data).items():  
 setattr(db\_professor, attr, value)  
 db.commit() # ثبت تراکنش برای ذخیره تغییرات  
 db.refresh(db\_professor) # به روزرسانی شی استاد برای دریافت مقادیر به‌روز شده  
 return db\_professor # بازگشت شی استاد به روزرسانی شده  
 raise HTTPException(status\_code=404, detail="lid not found")  
  
  
  
  
def delete\_professor(db: Session, professor\_lid: str):  
 *"""  
 حذف یک استاد براساس کد استاد  
  
 :param db: نشست دیتابیس  
 :param professor\_lid: کد استاد  
 :return: پاسخ JSON موفقیت آمیز در صورت حذف موفقیت آمیز  
 """* db\_professor = db.query(models.Professor).filter(models.Professor.lid == professor\_lid).first()  
 if db\_professor:  
 db.delete(db\_professor) # حذف شی استاد از نشست  
 db.commit() # ثبت تراکنش برای اعمال تغییرات در پایگاه داده  
 return db\_professor # بازگرداندن شی استاد حذف شده  
 return None # اگر استاد پیدا نشد بازگرداندن None  
  
  
  
  
# COURSE  
  
def create\_course(db: Session, course: schemas.Course):  
 *"""  
 ایجاد یک درس جدید در پایگاه داده  
  
 :param db: نشست دیتابیس  
 :param course: اطلاعات درس برای ثبت  
 :return: شی درس ایجاد شده  
 """* db\_course = models.Course(  
 cid=course.cid,  
 cname=course.cname,  
 department=course.department,  
 credit=course.credit  
 )  
 db.add(db\_course) # افزودن درس به نشست  
 db.commit() # ثبت تراکنش در پایگاه داده  
 db.refresh(db\_course) # به‌روزرسانی شی درس برای دریافت مقادیر به‌روز شده  
 return db\_course # بازگشت شی درس ایجاد شده  
  
  
def get\_course(db: Session, course\_cid: str) -> models.Course:  
 *"""  
 دریافت یک درس براساس کد درسی  
  
 :param db: نشست دیتابیس  
 :param course\_cid: کد درسی  
 :return: شی درس مطابق با کد درسی داده شده  
 """* return db.query(models.Course).filter(models.Course.cid == course\_cid).first()  
  
  
def update\_course(db: Session, course\_cid: str, new\_course\_data: schemas.Course):  
 *"""  
 به‌روزرسانی اطلاعات یک درس براساس کد درس  
  
 :param db: نشست دیتابیس  
 :param course\_cid: کد درسی  
 :param new\_course\_data: اطلاعات جدید درس برای به روزرسانی  
 :return: شی درس به روزرسانی شده  
 """* db\_course = db.query(models.Course).filter(models.Course.cid == course\_cid).first()  
 if db\_course:  
 # به روزرسانی هر ویژگی از درس با مقادیر جدید  
 for attr, value in vars(new\_course\_data).items():  
 setattr(db\_course, attr, value)  
 db.commit() # ثبت تراکنش برای ذخیره تغییرات  
 db.refresh(db\_course) # به روزرسانی شی درس برای دریافت مقادیر به روز شده  
 return db\_course # بازگشت شی درس به روزرسانی شده  
 raise HTTPException(status\_code=404, detail="cid not found")  
  
  
def delete\_course(db: Session, course\_cid: str):  
 *"""  
 حذف یک درس براساس کد درسی  
  
 :param db: نشست دیتابیس  
 :param course\_cid: کد درسی  
 :return: شی درس حذف شده  
 """* db\_course = db.query(models.Course).filter(models.Course.cid == course\_cid).first()  
 if db\_course:  
 db.delete(db\_course) # حذف شی درس از نشست  
 db.commit() # ثبت تراکنش برای اعمال تغییرات در پایگاه داده  
 return db\_course # بازگرداندن شی درس حذف شده  
 return None # اگر درس پیدا نشد بازگرداندن None

1. ***datavalidation.py***

در این فایل اعتبار سنجی ها رو مینویسیم. برای هر جدول یک کلاس میسازیم و توابع اعتبار سنجی آن جدول را درون کلاس قرار میدهیم

ابتدا نیازمندی ها را import میکنیم

# وارد کردن کلاس HTTPException از FastAPI برای استفاده در مدیریت خطاها  
from fastapi import HTTPException  
  
# وارد کردن کلاس Session از SQLAlchemy ORM برای استفاده در توابع CRUD  
from sqlalchemy.orm import Session  
  
# وارد کردن ماژول‌های crud، models و schemas از ماژول فعلی  
from . import crud, models, schemas

سپس یک کلاس برای اعتبار سنجی های جدول دانشجو درست میکنیم تا توابع اعتبار سنجی دانشجو را رد این کلاس قرار دهیم

سپس توابع اعتبار سنجی مربوط به جدول دانشجو را در این کلاس مینویسیم:

# STUDENT  
  
class student\_datavalidation:

# بررسی وجود دانشجو با استفاده از شماره دانشجویی  
 def student\_exists\_check(db: Session, stid: str):  
 student\_stid = db.query(models.Student).filter(models.Student.stid == stid).first()  
 if student\_stid is not None:  
 raise HTTPException(status\_code=409, detail="student with this stid already exists")

# بررسی صحت شماره دانشجویی  
 def check\_stid(db: Session, stid: str):  
 if len(stid) != 11:  
 raise HTTPException(status\_code=400, detail="Student ID must be 11 digits. The entered number of digits is incorrect.")  
 if int(stid[0:3]) > 403 or int(stid[0:3]) < 400:  
 raise HTTPException(status\_code=400, detail="Incorrect year part.")  
 if int(stid[3:9]) != 114150:  
 raise HTTPException(status\_code=400, detail="Incorrect fixed part.")  
 if int(stid[9:11]) == 0:  
 raise HTTPException(status\_code=400, detail="Incorrect index part.")

# بررسی صحت نام و نام خانوادگی و نام پدر  
 def check\_fflname(db: Session, fname: str, lname: str, father: str):  
 if len(fname) > 10 or len(lname) > 10 or len(father) > 10:  
 raise HTTPException(status\_code=400, detail="The number of names cannot be more than 10")  
  
 if not all('آ' <= char <= 'ی' or char == ' ' for char in (fname + lname + father)):  
 raise HTTPException(status\_code=400,  
 detail="The entered name should only contain Persian letters and"  
 " should not contain numbers, special characters")

# بررسی صحت تاریخ تولد  
 def check\_birth(db: Session, birth: str):  
 if len(birth) != 10:  
 raise HTTPException(status\_code=400, detail="The entered date must be exactly 10 digits"  
 " and separated by periods")  
  
 parts = birth.split('.')  
 if len(parts) != 3:  
 raise HTTPException(status\_code=400, detail="The date must have three parts"  
 " (year, month and day) respectively.")  
  
 year, month, day = parts  
 if not (year.isdigit() and month.isdigit() and day.isdigit()):  
 raise HTTPException(status\_code=400, detail="The date entered must contain numbers")  
  
 year = int(year)  
 month = int(month)  
 day = int(day)  
  
 if not (1300 <= year <= 1402):  
 raise HTTPException(status\_code=400, detail="The entered year must be between 1300 and 1402")  
  
 if not (1 <= month <= 12):  
 raise HTTPException(status\_code=400, detail="The month entered must be between 1 and 12")  
  
 if (1 <= month <= 6) and not (1 <= day <= 31):  
 raise HTTPException(status\_code=400, detail="The day entered must be between 1 and 31")  
  
 if (7 <= month <= 12) and not (1 <= day <= 30):  
 raise HTTPException(status\_code=400, detail="The day entered must be between 1 and 30")

# بررسی صحت شماره شناسنامه  
 def check\_ids(db: Session, ids: str):  
 parts = ids.split('.')  
 if len(parts) != 3:  
 raise HTTPException(status\_code=400, detail="Birth certificate serial number must consist of three parts")  
  
 first = parts[0]  
 second = parts[1]  
 third = parts[2]  
  
 if len(first) != 1:  
 raise HTTPException(status\_code=400, detail="The first part of the birth certificate serial number must consist of one letter")  
 if not ('آ' <= first <= 'ی'):  
 raise HTTPException(status\_code=400, detail="The first part of the birth certificate serial number must be one of the Persian alphabet letters")  
  
 if len(second) != 2:  
 raise HTTPException(status\_code=400, detail="The second part of the birth certificate serial number must consist of 2 digits")  
 if not second.isdigit():  
 raise HTTPException(status\_code=400, detail="The second part of the birth certificate serial number must be numeric")  
 if len(third) != 6:  
 raise HTTPException(status\_code=400, detail="The third part of the birth certificate serial number must consist of 6 digits")  
 if not third.isdigit():  
 raise HTTPException(status\_code=400, detail="The third part of the birth certificate serial number must be numeric")

# بررسی صحت شهر محل تولد مرکز استان  
 def born\_city(db: Session, born\_city: str):  
 markaz = ["اراک", "خرم آباد", "تهران", "اردبیل", "تبریز", "اصفهان", "اهواز", "ایلام", "بجنورد", "بیرجند", "مشهد",  
 "بندرعباس", "بوشهر", "ارومیه", "رشت", "زاهدان", "زنجان", "سمنان", "سنندج", "شیراز", "شهرکرد", "قزوین",  
 "قم", "کرج", "کرمان", "کرمانشاه", "گرگان", "همدان", "یاسوج", "یزد"]  
 if not (born\_city in markaz):  
 raise HTTPException(status\_code=400, detail="The entered province is not in the list of Iranian provinces"  
 " or the input was entered incorrectly")

# بررسی صحت آدرس  
 def check\_address(db: Session, address: str):  
 if len(address) > 100:  
 raise HTTPException(status\_code=400, detail="The length of the address exceeds the limit (100 characters).")

# بررسی صحت کد پستی  
 def check\_postalcode(db: Session, postalcode: int):  
 strpostalcode = str(postalcode)  
 if len(strpostalcode) != 10 or not strpostalcode.isdigit():  
 raise HTTPException(status\_code=400, detail="postalcode must be 10 digits")

# بررسی صحت تلفن همراه  
 def check\_cphone(db: Session, cphone: str):  
 if len(cphone) != 14 or not cphone[5:].isdigit():  
 raise HTTPException(status\_code=400, detail="The cphone number entered is not valid")  
 if cphone[0:4] != "+98-":  
 raise HTTPException(status\_code=400, detail="The cphone number must start with +98")

# بررسی صحت تلفن ثابت  
 def check\_hphone(db: Session, hphone: str):  
 if hphone.count('\_') != 1:  
 raise HTTPException(status\_code=400, detail="The entered hphone number must have two parts prefix"  
 " and fixed number and separated by \_")  
  
 parts = hphone.split('\_')  
 first = parts[0]  
 second = parts[1]  
  
 if len(first) != 3 or len(second) != 8:  
 raise HTTPException(status\_code=400, detail="The prefix must be three digits and"  
 " the hphone number must be eight digits")  
  
 if not (first.isdigit() and second.isdigit()):  
 raise HTTPException(status\_code=400, detail="The hphone number must be entered as a number")

# بررسی صحت دانشکده  
 def check\_department(db: Session, department: str):  
 valid\_dp = {  
 "فنی و مهندسی",  
 "علوم پایه",  
 "علوم انسانی",  
 "دامپزشکی",  
 "اقتصاد",  
 "کشاورزی",  
 "منابع طبیعی"  
 }  
  
 if department not in valid\_dp:  
 raise HTTPException(status\_code=400, detail="The department is invalid")

# بررسی صحت رشته تحصیلی  
 def check\_major(db: Session, major: str):  
 valid\_majors = ["مهندسی کامپیوتر", "مهندسی برق(الکترونیک)", "مهندسی برق(قدرت)", "مهندسی مکانیک و پلیمر",  
 "مهندسی معدن", "مهندسی عمران", "مهندسی شهرسازی"]  
 if major not in valid\_majors:  
 raise HTTPException(status\_code=400, detail="The major is invalid")

# بررسی وضعیت تاهل  
 def check\_married(db: Session, married: str):  
 if not (married == "مجرد" or married == "متاهل"):  
 raise HTTPException(status\_code=400, detail="Invalid marital status.")

# بررسی صحت کد ملی  
 def check\_id(db: Session, id: int):  
 student = db.query(models.Student).filter(models.Student.id == id).first()  
 if student is not None:  
 raise HTTPException(status\_code=400, detail="This ID already exists")  
  
 id\_str = str(id)  
 if len(id\_str) != 10 or not id\_str.isdigit():  
 raise HTTPException(status\_code=400, detail="The entered ID should be 10 digits")  
  
 check = int(id\_str[9])  
 sum = 0  
 for i in range(9):  
 sum += int(id\_str[i]) \* (10 - i)  
 c = sum % 11  
  
 if not (c < 2 and check == c or c >= 2 and check == 11 - c):  
 raise HTTPException(status\_code=400, detail="Invalid ID number")

کلاس و توابع اعتبار سنجی جدول استاد:

# PROFESSOR  
class professor\_datavalidation:  
  
 # بررسی وجود استاد با استفاده از کد استاد  
 def professor\_exists\_check(db: Session, lid: str):  
 professor\_lid = db.query(models.Professor).filter(models.Professor.lid == lid).first()  
 if professor\_lid is not None:  
 raise HTTPException(status\_code=409, detail="professor with this lid already exists")  
  
 # بررسی صحت کد استاد  
 def check\_lid(db: Session, lid: str):  
 if len(lid) != 6:  
 raise HTTPException(status\_code=400, detail="lid must be 6 characters")  
 if not lid.isdigit():  
 raise HTTPException(status\_code=400, detail="lid must be number")  
  
 # بررسی صحت نام و نام خانوادگی استاد  
 def check\_flname(db: Session, fname: str, lname: str):  
 if len(fname) > 11 or len(lname) > 11:  
 raise HTTPException(status\_code=400, detail="The number of names cannot be more than 10")  
  
 if not all('آ' <= char <= 'ی' or char == ' ' for char in (fname + lname)):  
 raise HTTPException(status\_code=400,  
 detail="The entered name should only contain Persian letters and should not contain numbers, special characters")  
  
 # بررسی صحت کد ملی استاد  
 def check\_id(db: Session, id: int):  
 professor = db.query(models.Professor).filter(models.Professor.id == id).first()  
 if professor is not None:  
 raise HTTPException(status\_code=400, detail="This ID already exists")  
  
 id\_str = str(id)  
 if len(id\_str) != 10 or not id\_str.isdigit():  
 raise HTTPException(status\_code=400, detail="The entered ID should be 10 digits")  
  
 check = int(id\_str[9])  
 sum = 0  
 for i in range(9):  
 sum += int(id\_str[i]) \* (10 - i)  
 c = sum % 11  
  
 if not (c < 2 and check == c or c >= 2 and check == 11 - c):  
 raise HTTPException(status\_code=400, detail="Invalid ID number")  
  
 # بررسی صحت دانشکده  
 def check\_department(db: Session, department: str):  
 valid\_dp = {  
 "فنی و مهندسی",  
 "علوم پایه",  
 "علوم انسانی",  
 "دامپزشکی",  
 "اقتصاد",  
 "کشاورزی",  
 "منابع طبیعی"  
 }  
  
 if department not in valid\_dp:  
 raise HTTPException(status\_code=400, detail="The department is invalid")  
  
 # بررسی صحت رشته تحصیلی  
 def check\_major(db: Session, major: str):  
 valid\_majors = ["مهندسی کامپیوتر", "مهندسی برق(الکترونیک)", "مهندسی برق(قدرت)", "مهندسی مکانیک و پلیمر",  
 "مهندسی معدن", "مهندسی عمران", "مهندسی شهرسازی"]  
 if major not in valid\_majors:  
 raise HTTPException(status\_code=400, detail="The major is invalid")  
  
 # بررسی صحت تاریخ تولد استاد  
 def check\_birth(db: Session, birth: str):  
 if len(birth) != 10:  
 raise HTTPException(status\_code=400, detail="The entered date must be exactly 10 digits"  
 " and separated by periods")  
  
 parts = birth.split('.')  
 if len(parts) != 3:  
 raise HTTPException(status\_code=400, detail="The date must have three parts"  
 " (year, month and day) respectively.")  
  
 year, month, day = parts  
 if not (year.isdigit() and month.isdigit() and day.isdigit()):  
 raise HTTPException(status\_code=400, detail="The date entered must contain numbers")  
  
 year = int(year)  
 month = int(month)  
 day = int(day)  
  
 if not (1300 <= year <= 1402):  
 raise HTTPException(status\_code=400, detail="The entered year must be between 1300 and 1402")  
  
 if not (1 <= month <= 12):  
 raise HTTPException(status\_code=400, detail="The month entered must be between 1 and 12")  
  
 if (1 <= month <= 6) and not (1 <= day <= 31):  
 raise HTTPException(status\_code=400, detail="The day entered must be between 1 and 31")  
  
 if (7 <= month <= 12) and not (1 <= day <= 30):  
 raise HTTPException(status\_code=400, detail="The day entered must be between 1 and 30")  
  
 # بررسی صحت شهر محل تولد مرکز استان استاد  
 def born\_city(db: Session, born\_city: str):  
 markaz = ["اراک", "خرم آباد", "تهران", "اردبیل", "تبریز", "اصفهان", "اهواز", "ایلام", "بجنورد", "بیرجند", "مشهد",  
 "بندرعباس", "بوشهر", "ارومیه", "رشت", "زاهدان", "زنجان", "سمنان", "سنندج", "شیراز", "شهرکرد", "قزوین",  
 "قم", "کرج", "کرمان", "کرمانشاه", "گرگان", "همدان", "یاسوج", "یزد"]  
 if not (born\_city in markaz):  
 raise HTTPException(status\_code=400, detail="The entered province is not in the list of Iranian provinces"  
 " or the input was entered incorrectly")  
  
 # بررسی صحت آدرس استاد  
 def check\_address(db: Session, address: str):  
 if len(address) > 100:  
 raise HTTPException(status\_code=400, detail="The length of the address exceeds the limit (100 characters).")  
  
 # بررسی صحت کد پستی استاد  
 def check\_postalcode(db: Session, postalcode: int):  
 strpostalcode = str(postalcode)  
 if len(strpostalcode) != 10 or not strpostalcode.isdigit():  
 raise HTTPException(status\_code=400, detail="postalcode must be 10 digits")  
  
 # بررسی صحت شماره تماس همراه استاد  
 def check\_cphone(db: Session, cphone: str):  
 if len(cphone) != 14 or not cphone[5:].isdigit():  
 raise HTTPException(status\_code=400, detail="The cphone number entered is not valid")  
 if cphone[0:4] != "+98-":  
 raise HTTPException(status\_code=400, detail="The cphone number must start with +98")  
  
 # بررسی صحت شماره تماس ثابت استاد  
 def check\_hphone(db: Session, hphone: str):  
 if hphone.count('\_') != 1:  
 raise HTTPException(status\_code=400, detail="The entered phone number must have two parts prefix"  
 " and fixed number and separated by \_")  
  
 parts = hphone.split('\_')  
 first = parts[0]  
 second = parts[1]  
  
 if len(first) != 3 or len(second) != 8:  
 raise HTTPException(status\_code=400, detail="The prefix must be three digits and"  
 " the landline number must be eight digits")  
  
 if not (first.isdigit() and second.isdigit()):  
 raise HTTPException(status\_code=400, detail="The phone number must be entered as a number")

کلاس و توابع اعتبار سنجی جدول درس:

#COURSE  
class course\_datavalidation:  
  
 # بررسی وجود درس با کد درس مشابه در دیتابیس  
 def course\_exists\_check(db: Session, cid: str):  
 course\_cid = db.query(models.Course).filter(models.Course.cid == cid).first()  
 if course\_cid is not None:  
 raise HTTPException(status\_code=409, detail="Course with this cid already exists")  
  
 # بررسی طول و نوع داده کد درس  
 def check\_cid(db: Session, cid: str):  
 if len(cid) != 5:  
 raise HTTPException(status\_code=400, detail="cid must be 5 characters")  
 if not cid.isdigit():  
 raise HTTPException(status\_code=400, detail="cid must be number")  
  
 # بررسی صحت نام درس  
 def check\_cname(db: Session, cname: str):  
 # بررسی طول نام دوره  
 if len(cname) > 25:  
 raise HTTPException(status\_code=400, detail="The number of characters cannot be more than 25")  
  
 # بررسی اعتبار حروف فارسی و فاصله در نام دوره  
 for char in cname:  
 if not ('آ' <= char <= 'ی' or char == ' '):  
 raise HTTPException(status\_code=400,  
 detail="The entered name should only contain Persian letters and spaces")  
  
 # بررسی صحت دانشکده  
 def check\_department\_course(db: Session, department: str):  
 # بررسی مقایسه دوره با دپارتمان های معتبر  
 valid\_dp = {  
 "فنی و مهندسی",  
 "علوم پایه",  
 "علوم انسانی",  
 "دامپزشکی",  
 "اقتصاد",  
 "کشاورزی",  
 "منابع طبیعی"  
 }  
  
 if department not in valid\_dp:  
 raise HTTPException(status\_code=400, detail="The department is invalid")  
  
 # بررسی صحت تعداد واحد درس   
 def check\_credit(db: Session, credit: int):  
 if not (1 <= credit <= 4):  
 raise HTTPException(status\_code=400, detail="Credit must be an integer between 1 and 4")

1. ***main.py:***

فایل main شامل تمامی اطلاعات و موجودیت های دیتابیس است و از تمامی عملیات CRUD پشتیبانی می کند تا به راحتی بتوان داده ها را مدیریت کرد و اطلاعات مورد نیاز را از دیتابیس خواند یا به دیتابیس اضافه کرد یا آنها را از دیتابیس حذف کرد

توضیحات کلی:

این کد از FastAPI و SQLAlchemy ORM برای انجام عملیات CRUD بر روی سه جدول موجود در دیتابیس استفاده میکند. دسترسی به این API از طریق مسیرهای مختلفی که بر اساس نوع عملیات و شناسه مورد نظر تعریف شده اند، امکان پذیر است

اجزای اصلی کد:

models:

شامل تعریف جداول دیتابیس به کمک SQLAlchemy.

schemas:

تعریف داده های ورودی و خروجی API که با استفاده از Pydantic اعتبارسنجی می‌شوند و کمک میکنند فقط 4 فیلد اول در خروجی نمایش داده شوند

crud:

توابعی که عملیات ایجاد (Create)، خواندن (Read)، به‌روزرسانی (Update) و حذف (Delete) را بر روی دیتابیس انجام می‌دهند.

database:

ایجاد اتصال به دیتابیس SQLite و تولید چیزهای مورد نیاز برای استفاده در توابع CRUD

datavalidation:

توابعی که برای اعتبارسنجی داده های ورودی قبل از ارسال به دیتابیس تعریف شده اند تا اطمینان حاصل شود که داده های ورودی معتبر و مناسب هستند

FastAPI:

ایجاد یک نمونه از FastAPI و تعریف مسیرها و توابع مربوط به هر جدول (دانشجو، استاد، درس).

عملیات های اصلی:

POST: برای ایجاد دانشجو، استاد و درس با کمک اعتبارسنجی بر روی داده های ورودی

GET : برای خواندن اطلاعات دانشجو، استاد و درس بر اساس شناسه هایشان

PUT : برای به روزرسانی اطلاعات دانشجو، استاد و درس با کمک اعتبارسنجی داده های جدید

DELETE : برای حذف دانشجو، استاد و درس بر اساس شناسه هایشان

ابتدا نیازمندی هایمان را Import میکنیم

# وارد کردن متغیرهای لازم از FastAPI و SQLAlchemy ORM  
from fastapi import Depends, FastAPI, HTTPException  
  
# وارد کردن کلاس Session از SQLAlchemy ORM برای استفاده در توابع CRUD  
from sqlalchemy.orm import Session  
  
# وارد کردن کلاس‌ها و توابع CRUD از بخشهای مختلف پروژه  
from . import crud, models, schemas  
  
# وارد کردن اتصال موقت به دیتابیس و مهیا کردن متغیرهای مربوطه  
from .database import SessionLocal, engine  
  
# وارد کردن توابع اعتبارسنجی اطلاعات برای دوره، استاد و دانشجو  
from .datavalidation import course\_datavalidation, professor\_datavalidation, student\_datavalidation

سپس یک تابع تعریف میکنیم که ما را موقتا به دیتابیس وصل کند و پس از استفاده بسته شود

برای اینکار تابع SessionLocal() را صدا میزنیم که یک اتصال به دیتابیس برمی گرداند و آن را در db ذخیره میکند

سپس تابع با استفاده از yield db ، db را بعنوان خروجی برمیگرداند

در آخر با استفاده از finally و db,close() اتصال به دیتابیس بسته میشود تا منابع دیتابیس آزاد شوند و از حافظه استفاده بهینه شود

# تابع برای تأمین اتصال به دیتابیس به صورت موقت و بستن آن پس از استفاده  
def get\_db():  
 db = SessionLocal()  
 try:  
 yield db  
 finally:  
 db.close()

قسمت create و پست:

خط اول نوع درخواست که پست است و آدرس مسیر را تعیین میکند و response\_model=schemas.Student\_set مشخص میکند که خروجی باید با schemas.Student\_set مطابقت داشته باشد و بر این اساس خروجی ها به 4 فیلد اول محدود میشوند

سپس یک تابع برای ساختن اطلاعات تعریف میشود که شامل دو ورودی است

ورودی اول student است که شامل schemas.Student میباشد که این حاوی اطلاعات دانشجو است

ورودی دوم db: Session = Depends(get\_db) است که اتصال به دیتابیس را برقرار میکند

@app.post("/create\_student/", response\_model=schemas.Student\_set)  
def create\_student(student: schemas.Student, db: Session = Depends(get\_db)):

قبل از ایجاد دانشجو، تمام اطلاعات ورودی با کمک اعتبار سنجی هایی که از فایل datavalidation.py گرفته شده اند چک میشوند تا درستی و معتبر بودنشان تایید شود و در غیر این صورت پیغام خطای مناسب نشان داده شود

student\_datavalidation.student\_exists\_check(db, student.stid)  
student\_datavalidation.check\_stid(db, student.stid)  
student\_datavalidation.check\_fflname(db, student.fname, student.lname, student.father)  
student\_datavalidation.check\_birth(db, student.birth)  
student\_datavalidation.check\_ids(db, student.ids)  
student\_datavalidation.born\_city(db, student.born\_city)  
student\_datavalidation.check\_address(db, student.address)  
student\_datavalidation.check\_postalcode(db, student.postalcode)  
student\_datavalidation.check\_cphone(db, student.cphone)  
student\_datavalidation.check\_hphone(db, student.hphone)  
student\_datavalidation.check\_department(db, student.department)  
student\_datavalidation.check\_major(db, student.major)  
student\_datavalidation.check\_married(db, student.married)  
student\_datavalidation.check\_id(db, student.id)

سپس اطلاعات دانشجو به عنوان یک شی از نوع schemas.Student به تابعcrud.create\_student که مسئول ذخیره اطلاعات دانشجو در دیتابیس است ارسال میشود و در پاسخ برگردانده میشود

# ایجاد دانشجو با استفاده از دیتابیس  
return crud.create\_student(db=db, student=student)

کلش:

@app.post("/create\_student/", response\_model=schemas.Student\_set)  
def create\_student(student: schemas.Student, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای ایجاد دانشجو  
 student\_datavalidation.student\_exists\_check(db, student.stid)  
 student\_datavalidation.check\_stid(db, student.stid)  
 student\_datavalidation.check\_fflname(db, student.fname, student.lname, student.father)  
 student\_datavalidation.check\_birth(db, student.birth)  
 student\_datavalidation.check\_ids(db, student.ids)  
 student\_datavalidation.born\_city(db, student.born\_city)  
 student\_datavalidation.check\_address(db, student.address)  
 student\_datavalidation.check\_postalcode(db, student.postalcode)  
 student\_datavalidation.check\_cphone(db, student.cphone)  
 student\_datavalidation.check\_hphone(db, student.hphone)  
 student\_datavalidation.check\_department(db, student.department)  
 student\_datavalidation.check\_major(db, student.major)  
 student\_datavalidation.check\_married(db, student.married)  
 student\_datavalidation.check\_id(db, student.id)  
  
 # ایجاد دانشجو با استفاده از دیتابیس  
 return crud.create\_student(db=db, student=student)

قسمت get و خواندن:

تعریف دو خط اول این قسمت مانند قسمت قبل است. در قسمت بعدی اطلاعات از دیتابیس بر اساس شماره دانشجویی خوانده میشود. اگر شماره دانشجویی پیدا نشد پیام خطا نشان داده میشود ولی اگر یافت شد اطلاعات در خروجی نمایش داده میشوند ولی به دلیل قسمت schemas.Student\_set فقط 4 فیلد اول به کاربر نشان داده میشوند

@app.get("/get\_student/{student\_stid}", response\_model=schemas.Student\_set)  
def read\_student(student\_stid: str, db: Session = Depends(get\_db)):  
 # خواندن اطلاعات دانشجو بر اساس شماره دانشجویی  
 db\_student = crud.get\_student(db, student\_stid=student\_stid)  
 if db\_student is None:  
 raise HTTPException(status\_code=404, detail="Student not found")  
 return db\_student

قسمت update(put) یا به روزرسانی:

خط اول و دوم این کد مانند کد قبل است با این فرق که در خط دوم یه فیلد دیگر به ورودی تابع اضافه میشود این فیلد از نوع schemas.Student است که شامل اطلاعات جدید برای به روزرسانی دانشجو می باشد و سپس بخش های اعتبار سنجی قرار میگیرند

در آخر تابع آپدیت از فایل crud فراخوانی میشود تا اطلاعات جدید را با قبلی جایگزین کند

این به روز رسانی ها در db\_student ذخیره میشوند و در آخر برگردانده میشود

@app.put("/update\_student/{student\_stid}", response\_model=schemas.Student\_set)  
def update\_student(student\_stid: str, new\_data: schemas.Student, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای به‌روزرسانی اطلاعات دانشجو  
 student\_datavalidation.check\_stid(db, new\_data.stid)  
 student\_datavalidation.check\_fflname(db, new\_data.fname, new\_data.lname, new\_data.father)  
 student\_datavalidation.check\_birth(db, new\_data.birth)  
 student\_datavalidation.check\_ids(db, new\_data.ids)  
 student\_datavalidation.born\_city(db, new\_data.born\_city)  
 student\_datavalidation.check\_address(db, new\_data.address)  
 student\_datavalidation.check\_postalcode(db, new\_data.postalcode)  
 student\_datavalidation.check\_cphone(db, new\_data.cphone)  
 student\_datavalidation.check\_hphone(db, new\_data.hphone)  
 student\_datavalidation.check\_department(db, new\_data.department)  
 student\_datavalidation.check\_major(db, new\_data.major)  
 student\_datavalidation.check\_married(db, new\_data.married)  
 student\_datavalidation.check\_id(db, new\_data.id)  
  
 # به‌روزرسانی اطلاعات دانشجو در دیتابیس  
 db\_student = crud.update\_student(db, student\_stid=student\_stid, new\_student\_data=new\_data)  
 return db\_student

قسمت delete یا حذف:

در این کد هم خط اول و دوم مانند قبلی هاست

سپس تابع crud.delete\_student فراخوانی میشود تا دانشجو با شماره دانشجویی مشخص شده را از دیتابیس حذف کند و نتیجه در deleted\_student ذخیره میشود

اگه دانشجو با موفقیت حذف شده بود پیام مربوطه به کاربر نشان داده میشود ولی اگر دانشجویی پیدا نشد پیغام خطای مربوطه نشان داده میشود

@app.delete("/delete\_student/{student\_stid}")  
def delete\_student(student\_stid: str, db: Session = Depends(get\_db)):  
 # حذف دانشجو بر اساس شماره دانشجویی  
 deleted\_student = crud.delete\_student(db, student\_stid=student\_stid)  
 if deleted\_student:  
 return {"message": "Delete successful"}  
 # اگر دانشجو یافت نشد، خطای مناسب با کد 404 بازگردانده می‌شود  
 raise HTTPException(status\_code=404, detail="Student not found")

کد کلی دانشجو:

# STUDENT  
  
@app.post("/create\_student/", response\_model=schemas.Student\_set)  
def create\_student(student: schemas.Student, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای ایجاد دانشجو  
 student\_datavalidation.student\_exists\_check(db, student.stid)  
 student\_datavalidation.check\_stid(db, student.stid)  
 student\_datavalidation.check\_fflname(db, student.fname, student.lname, student.father)  
 student\_datavalidation.check\_birth(db, student.birth)  
 student\_datavalidation.check\_ids(db, student.ids)  
 student\_datavalidation.born\_city(db, student.born\_city)  
 student\_datavalidation.check\_address(db, student.address)  
 student\_datavalidation.check\_postalcode(db, student.postalcode)  
 student\_datavalidation.check\_cphone(db, student.cphone)  
 student\_datavalidation.check\_hphone(db, student.hphone)  
 student\_datavalidation.check\_department(db, student.department)  
 student\_datavalidation.check\_major(db, student.major)  
 student\_datavalidation.check\_married(db, student.married)  
 student\_datavalidation.check\_id(db, student.id)  
  
 # ایجاد دانشجو با استفاده از دیتابیس  
 return crud.create\_student(db=db, student=student)  
  
@app.get("/get\_student/{student\_stid}", response\_model=schemas.Student\_set)  
def read\_student(student\_stid: str, db: Session = Depends(get\_db)):  
 # خواندن اطلاعات دانشجو بر اساس شماره دانشجویی  
 db\_student = crud.get\_student(db, student\_stid=student\_stid)  
 if db\_student is None:  
 raise HTTPException(status\_code=404, detail="Student not found")  
 return db\_student  
  
  
  
@app.put("/update\_student/{student\_stid}", response\_model=schemas.Student\_set)  
def update\_student(student\_stid: str, new\_data: schemas.Student, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای به‌روزرسانی اطلاعات دانشجو  
 student\_datavalidation.check\_stid(db, new\_data.stid)  
 student\_datavalidation.check\_fflname(db, new\_data.fname, new\_data.lname, new\_data.father)  
 student\_datavalidation.check\_birth(db, new\_data.birth)  
 student\_datavalidation.check\_ids(db, new\_data.ids)  
 student\_datavalidation.born\_city(db, new\_data.born\_city)  
 student\_datavalidation.check\_address(db, new\_data.address)  
 student\_datavalidation.check\_postalcode(db, new\_data.postalcode)  
 student\_datavalidation.check\_cphone(db, new\_data.cphone)  
 student\_datavalidation.check\_hphone(db, new\_data.hphone)  
 student\_datavalidation.check\_department(db, new\_data.department)  
 student\_datavalidation.check\_major(db, new\_data.major)  
 student\_datavalidation.check\_married(db, new\_data.married)  
 student\_datavalidation.check\_id(db, new\_data.id)  
  
 # به‌روزرسانی اطلاعات دانشجو در دیتابیس  
 db\_student = crud.update\_student(db, student\_stid=student\_stid, new\_student\_data=new\_data)  
 return db\_student  
  
  
  
@app.delete("/delete\_student/{student\_stid}")  
def delete\_student(student\_stid: str, db: Session = Depends(get\_db)):  
 # حذف دانشجو بر اساس شماره دانشجویی  
 deleted\_student = crud.delete\_student(db, student\_stid=student\_stid)  
 if deleted\_student:  
 return {"message": "Delete successful"}  
 # اگر دانشجو یافت نشد، خطای مناسب با کد 404 بازگردانده می‌شود  
 raise HTTPException(status\_code=404, detail="Student not found")

همین مراحل بالا برای جدول های استاد و درس هم انجام میشوند

استاد:

# PROFESSOR  
  
@app.post("/create\_professor/", response\_model=schemas.Professor\_set)  
def create\_professor(professor: schemas.Professor, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای ایجاد استاد  
 professor\_datavalidation.professor\_exists\_check(db, professor.lid)  
 professor\_datavalidation.check\_lid(db, professor.lid)  
 professor\_datavalidation.check\_flname(db, professor.fname, professor.lname)  
 professor\_datavalidation.check\_id(db, professor.id)  
 professor\_datavalidation.check\_department(db, professor.department)  
 professor\_datavalidation.check\_major(db, professor.major)  
 professor\_datavalidation.check\_birth(db, professor.birth)  
 professor\_datavalidation.born\_city(db, professor.born\_city)  
 professor\_datavalidation.check\_address(db, professor.address)  
 professor\_datavalidation.check\_postalcode(db, professor.postalcode)  
 professor\_datavalidation.check\_cphone(db, professor.cphone)  
 professor\_datavalidation.check\_hphone(db, professor.hphone)  
  
 # ایجاد استاد با استفاده از دیتابیس  
 return crud.create\_professor(db=db, professor=professor)  
  
  
@app.get("/get\_professor/{professor\_lid}", response\_model=schemas.Professor\_set)  
def read\_professor(professor\_lid: str, db: Session = Depends(get\_db)):  
 # خواندن اطلاعات استاد بر اساس شناسه  
 db\_professor = crud.get\_professor(db, professor\_lid=professor\_lid)  
 if db\_professor is None:  
 # اگر استاد یافت نشد، خطای مناسب با کد 404 بازگردانده می‌شود  
 raise HTTPException(status\_code=404, detail="Professor not found")  
 return db\_professor  
  
  
  
@app.put("/update\_professor/{professor\_lid}", response\_model=schemas.Professor\_set)  
def update\_professor(professor\_lid: str, new\_data: schemas.Professor, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای به‌روزرسانی اطلاعات استاد  
 professor\_datavalidation.check\_lid(db, new\_data.lid)  
 professor\_datavalidation.check\_flname(db, new\_data.fname, new\_data.lname)  
 professor\_datavalidation.check\_id(db, new\_data.id)  
 professor\_datavalidation.check\_department(db, new\_data.department)  
 professor\_datavalidation.check\_major(db, new\_data.major)  
 professor\_datavalidation.check\_birth(db, new\_data.birth)  
 professor\_datavalidation.born\_city(db, new\_data.born\_city)  
 professor\_datavalidation.check\_address(db, new\_data.address)  
 professor\_datavalidation.check\_postalcode(db, new\_data.postalcode)  
 professor\_datavalidation.check\_cphone(db, new\_data.cphone)  
 professor\_datavalidation.check\_hphone(db, new\_data.hphone)  
  
 # به‌روزرسانی اطلاعات استاد در دیتابیس  
 db\_professor = crud.update\_professor(db, professor\_lid=professor\_lid, new\_professor\_data=new\_data)  
 return db\_professor  
  
  
@app.delete("/delete\_professor/{professor\_lid}")  
def delete\_professor(professor\_lid: str, db: Session = Depends(get\_db)):  
 # حذف استاد بر اساس شناسه  
 deleted\_professor = crud.delete\_professor(db, professor\_lid=professor\_lid)  
 if deleted\_professor:  
 return {"message": "Delete successful"}  
 raise HTTPException(status\_code=404, detail="Professor not found")

درس:

# COURSE  
  
@app.post("/create\_course/", response\_model=schemas.Course)  
def create\_course(course: schemas.Course, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای ایجاد دوره  
 course\_datavalidation.course\_exists\_check(db, course.cid)  
 course\_datavalidation.check\_cid(db, course.cid)  
 course\_datavalidation.check\_cname(db, course.cname)  
 course\_datavalidation.check\_department\_course(db, course.department)  
 course\_datavalidation.check\_credit(db, course.credit)  
  
 # ایجاد دوره با استفاده از دیتابیس  
 return crud.create\_course(db=db, course=course)  
  
@app.get("/get\_course/{course\_cid}", response\_model=schemas.Course)  
def read\_course(course\_cid: int, db: Session = Depends(get\_db)):  
 # خواندن اطلاعات دوره بر اساس شناسه  
 db\_course = crud.get\_course(db, course\_cid=course\_cid)  
 if db\_course is None:  
 # اگر دوره یافت نشد، خطای مناسب با کد 404 بازگردانده می‌شود  
 raise HTTPException(status\_code=404, detail="Course not found")  
 return db\_course  
  
@app.put("/update\_course/{course\_cid}", response\_model=schemas.Course)  
def update\_course(course\_cid: int, new\_data: schemas.Course, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای به‌روزرسانی اطلاعات دوره  
 course\_datavalidation.check\_cid(db, new\_data.cid)  
 course\_datavalidation.check\_cname(db, new\_data.cname)  
 course\_datavalidation.check\_department\_course(db, new\_data.department)  
 course\_datavalidation.check\_credit(db, new\_data.credit)  
  
 # به‌روزرسانی اطلاعات دوره در دیتابیس  
 db\_course = crud.update\_course(db, course\_cid=course\_cid, new\_course\_data=new\_data)  
 return db\_course  
  
  
  
@app.delete("/delete\_course/{course\_cid}")  
def delete\_course(course\_cid: str, db: Session = Depends(get\_db)):  
 *"""  
 حذف یک درس بر اساس کد درسی  
  
 :param course\_cid: کد درسی  
 :param db: نشست دیتابیس  
 :return: JSONResponse موفقیت آمیز در صورت حذف موفقیت آمیز  
 """* deleted\_course = crud.delete\_course(db, course\_cid=course\_cid)  
 if deleted\_course:  
 return {"message": "Delete successful"}  
 raise HTTPException(status\_code=404, detail="Course not found") # اگر دوره یافت نشد، خطای مناسب با کد 404 بازگردانده می‌شود

کل کد فایل main:

# from typing import List  
# وارد کردن متغیرهای لازم از FastAPI و SQLAlchemy ORM  
from fastapi import Depends, FastAPI, HTTPException  
  
# وارد کردن کلاس Session از SQLAlchemy ORM برای استفاده در توابع CRUD  
from sqlalchemy.orm import Session  
  
# وارد کردن کلاس‌ها و توابع CRUD از بخشهای مختلف پروژه  
from . import crud, models, schemas  
  
# وارد کردن اتصال موقت به دیتابیس و مهیا کردن متغیرهای مربوطه  
from .database import SessionLocal, engine  
  
# وارد کردن توابع اعتبارسنجی اطلاعات برای دوره، استاد و دانشجو  
from .datavalidation import course\_datavalidation, professor\_datavalidation, student\_datavalidation  
  
# ایجاد جدول‌های دیتابیس اس‌کیو‌ال بر اساس مدل‌های تعریف شده  
models.Base.metadata.create\_all(bind=engine)  
  
# ایجاد اپلیکیشن FastAPI  
app = FastAPI()  
  
# تابع برای تأمین اتصال به دیتابیس به صورت موقت و بستن آن پس از استفاده  
def get\_db():  
 db = SessionLocal()  
 try:  
 yield db  
 finally:  
 db.close()  
  
# STUDENT  
  
@app.post("/create\_student/", response\_model=schemas.Student\_set)  
def create\_student(student: schemas.Student, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای ایجاد دانشجو  
 student\_datavalidation.student\_exists\_check(db, student.stid)  
 student\_datavalidation.check\_stid(db, student.stid)  
 student\_datavalidation.check\_fflname(db, student.fname, student.lname, student.father)  
 student\_datavalidation.check\_birth(db, student.birth)  
 student\_datavalidation.check\_ids(db, student.ids)  
 student\_datavalidation.born\_city(db, student.born\_city)  
 student\_datavalidation.check\_address(db, student.address)  
 student\_datavalidation.check\_postalcode(db, student.postalcode)  
 student\_datavalidation.check\_cphone(db, student.cphone)  
 student\_datavalidation.check\_hphone(db, student.hphone)  
 student\_datavalidation.check\_department(db, student.department)  
 student\_datavalidation.check\_major(db, student.major)  
 student\_datavalidation.check\_married(db, student.married)  
 student\_datavalidation.check\_id(db, student.id)  
  
 # ایجاد دانشجو با استفاده از دیتابیس  
 return crud.create\_student(db=db, student=student)  
  
@app.get("/get\_student/{student\_stid}", response\_model=schemas.Student\_set)  
def read\_student(student\_stid: str, db: Session = Depends(get\_db)):  
 # خواندن اطلاعات دانشجو بر اساس شماره دانشجویی  
 db\_student = crud.get\_student(db, student\_stid=student\_stid)  
 if db\_student is None:  
 raise HTTPException(status\_code=404, detail="Student not found")  
 return db\_student  
  
  
  
@app.put("/update\_student/{student\_stid}", response\_model=schemas.Student\_set)  
def update\_student(student\_stid: str, new\_data: schemas.Student, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای به‌روزرسانی اطلاعات دانشجو  
 student\_datavalidation.check\_stid(db, new\_data.stid)  
 student\_datavalidation.check\_fflname(db, new\_data.fname, new\_data.lname, new\_data.father)  
 student\_datavalidation.check\_birth(db, new\_data.birth)  
 student\_datavalidation.check\_ids(db, new\_data.ids)  
 student\_datavalidation.born\_city(db, new\_data.born\_city)  
 student\_datavalidation.check\_address(db, new\_data.address)  
 student\_datavalidation.check\_postalcode(db, new\_data.postalcode)  
 student\_datavalidation.check\_cphone(db, new\_data.cphone)  
 student\_datavalidation.check\_hphone(db, new\_data.hphone)  
 student\_datavalidation.check\_department(db, new\_data.department)  
 student\_datavalidation.check\_major(db, new\_data.major)  
 student\_datavalidation.check\_married(db, new\_data.married)  
 student\_datavalidation.check\_id(db, new\_data.id)  
  
 # به‌روزرسانی اطلاعات دانشجو در دیتابیس  
 db\_student = crud.update\_student(db, student\_stid=student\_stid, new\_student\_data=new\_data)  
 return db\_student  
  
  
  
@app.delete("/delete\_student/{student\_stid}")  
def delete\_student(student\_stid: str, db: Session = Depends(get\_db)):  
 # حذف دانشجو بر اساس شماره دانشجویی  
 deleted\_student = crud.delete\_student(db, student\_stid=student\_stid)  
 if deleted\_student:  
 return {"message": "Delete successful"}  
 # اگر دانشجو یافت نشد، خطای مناسب با کد 404 بازگردانده می‌شود  
 raise HTTPException(status\_code=404, detail="Student not found")  
  
  
  
  
# PROFESSOR  
  
@app.post("/create\_professor/", response\_model=schemas.Professor\_set)  
def create\_professor(professor: schemas.Professor, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای ایجاد استاد  
 professor\_datavalidation.professor\_exists\_check(db, professor.lid)  
 professor\_datavalidation.check\_lid(db, professor.lid)  
 professor\_datavalidation.check\_flname(db, professor.fname, professor.lname)  
 professor\_datavalidation.check\_id(db, professor.id)  
 professor\_datavalidation.check\_department(db, professor.department)  
 professor\_datavalidation.check\_major(db, professor.major)  
 professor\_datavalidation.check\_birth(db, professor.birth)  
 professor\_datavalidation.born\_city(db, professor.born\_city)  
 professor\_datavalidation.check\_address(db, professor.address)  
 professor\_datavalidation.check\_postalcode(db, professor.postalcode)  
 professor\_datavalidation.check\_cphone(db, professor.cphone)  
 professor\_datavalidation.check\_hphone(db, professor.hphone)  
  
 # ایجاد استاد با استفاده از دیتابیس  
 return crud.create\_professor(db=db, professor=professor)  
  
  
@app.get("/get\_professor/{professor\_lid}", response\_model=schemas.Professor\_set)  
def read\_professor(professor\_lid: str, db: Session = Depends(get\_db)):  
 # خواندن اطلاعات استاد بر اساس شناسه  
 db\_professor = crud.get\_professor(db, professor\_lid=professor\_lid)  
 if db\_professor is None:  
 # اگر استاد یافت نشد، خطای مناسب با کد 404 بازگردانده می‌شود  
 raise HTTPException(status\_code=404, detail="Professor not found")  
 return db\_professor  
  
  
  
@app.put("/update\_professor/{professor\_lid}", response\_model=schemas.Professor\_set)  
def update\_professor(professor\_lid: str, new\_data: schemas.Professor, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای به‌روزرسانی اطلاعات استاد  
 professor\_datavalidation.check\_lid(db, new\_data.lid)  
 professor\_datavalidation.check\_flname(db, new\_data.fname, new\_data.lname)  
 professor\_datavalidation.check\_id(db, new\_data.id)  
 professor\_datavalidation.check\_department(db, new\_data.department)  
 professor\_datavalidation.check\_major(db, new\_data.major)  
 professor\_datavalidation.check\_birth(db, new\_data.birth)  
 professor\_datavalidation.born\_city(db, new\_data.born\_city)  
 professor\_datavalidation.check\_address(db, new\_data.address)  
 professor\_datavalidation.check\_postalcode(db, new\_data.postalcode)  
 professor\_datavalidation.check\_cphone(db, new\_data.cphone)  
 professor\_datavalidation.check\_hphone(db, new\_data.hphone)  
  
 # به‌روزرسانی اطلاعات استاد در دیتابیس  
 db\_professor = crud.update\_professor(db, professor\_lid=professor\_lid, new\_professor\_data=new\_data)  
 return db\_professor  
  
  
@app.delete("/delete\_professor/{professor\_lid}")  
def delete\_professor(professor\_lid: str, db: Session = Depends(get\_db)):  
 # حذف استاد بر اساس شناسه  
 deleted\_professor = crud.delete\_professor(db, professor\_lid=professor\_lid)  
 if deleted\_professor:  
 return {"message": "Delete successful"}  
 raise HTTPException(status\_code=404, detail="Professor not found")  
  
  
  
  
  
# COURSE  
  
@app.post("/create\_course/", response\_model=schemas.Course)  
def create\_course(course: schemas.Course, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای ایجاد دوره  
 course\_datavalidation.course\_exists\_check(db, course.cid)  
 course\_datavalidation.check\_cid(db, course.cid)  
 course\_datavalidation.check\_cname(db, course.cname)  
 course\_datavalidation.check\_department\_course(db, course.department)  
 course\_datavalidation.check\_credit(db, course.credit)  
  
 # ایجاد دوره با استفاده از دیتابیس  
 return crud.create\_course(db=db, course=course)  
  
@app.get("/get\_course/{course\_cid}", response\_model=schemas.Course)  
def read\_course(course\_cid: int, db: Session = Depends(get\_db)):  
 # خواندن اطلاعات دوره بر اساس شناسه  
 db\_course = crud.get\_course(db, course\_cid=course\_cid)  
 if db\_course is None:  
 # اگر دوره یافت نشد، خطای مناسب با کد 404 بازگردانده می‌شود  
 raise HTTPException(status\_code=404, detail="Course not found")  
 return db\_course  
  
@app.put("/update\_course/{course\_cid}", response\_model=schemas.Course)  
def update\_course(course\_cid: int, new\_data: schemas.Course, db: Session = Depends(get\_db)):  
 # اعتبارسنجی‌های مختلف برای به‌روزرسانی اطلاعات دوره  
 course\_datavalidation.check\_cid(db, new\_data.cid)  
 course\_datavalidation.check\_cname(db, new\_data.cname)  
 course\_datavalidation.check\_department\_course(db, new\_data.department)  
 course\_datavalidation.check\_credit(db, new\_data.credit)  
  
 # به‌روزرسانی اطلاعات دوره در دیتابیس  
 db\_course = crud.update\_course(db, course\_cid=course\_cid, new\_course\_data=new\_data)  
 return db\_course  
  
  
  
@app.delete("/delete\_course/{course\_cid}")  
def delete\_course(course\_cid: str, db: Session = Depends(get\_db)):  
 *"""  
 حذف یک درس بر اساس کد درسی  
  
 :param course\_cid: کد درسی  
 :param db: نشست دیتابیس  
 :return: JSONResponse موفقیت آمیز در صورت حذف موفقیت آمیز  
 """* deleted\_course = crud.delete\_course(db, course\_cid=course\_cid)  
 if deleted\_course:  
 return {"message": "Delete successful"}  
 raise HTTPException(status\_code=404, detail="Course not found") # اگر دوره یافت نشد، خطای مناسب با کد 404 بازگردانده می‌شود

داکر فایل:

دستور اول مشخص میکند که از ایمیج python:3.12.3 به عنوان پایه برای ساخت این Docker image استفاده می‌شود (اون عددای جلوی پایتون ورژنش هستن)

FROM python:3.12.3

دستور WORKDIR مشخص میکند که دایرکتوری فعلی کار در داخل Docker container را به /var/www تغییر میدهد. این دایرکتوری به عنوان محل کار برای اجرای دستورات بعدی در Docker container استفاده میشود

WORKDIR /var/www

دستورCOPY فایل requirements.txt را از مسیر sql\_app/ به مسیر فعلی در Docker container (که با WORKDIR تعیین شده است) کپی میکند

COPY sql\_app/requirements.txt .

با دستور RUN با استفاده از pip، تمام ورژن های موجود در فایل requirements.txt نصب میشوند

RUN pip install -r requirements.txt

دستور COPY تمام فایل ها و دایرکتوری های موجود در sql\_app را بدر محل فعلی که در WORKDIR مشخص کردیم در Docker container کپی می کند

COPY sql\_app .

دستور CMD دستور اصلی است که برنامه در Docker container با آن اجرا میشود. در اینجا، دستور fastapi و run و main.py را به عنوان CMD تعیین کرده ایم که FastAPI را برای اجرا از فایل main.py شروع میکند

CMD ["fastapi", "run", "main.py"]

کل کد:

FROM python:3.12.3  
  
WORKDIR /var/www  
  
COPY sql\_app/requirements.txt .  
  
  
RUN pip install -r requirements.txt  
  
  
COPY sql\_app .  
  
CMD ["fastapi", "run", "main.py"]