به نام خالق

برای ایجاد پروژه در مرحله اول بایستی (database) تهیه شود. برای ایجاد ارتباط بین دیتا بیس خودمان و (sqlalchemy) لازم است ابتدا یک انجین داشته باشیم:

```python
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

SQLALCHEMY_DATABASE_URL = "sqlite:///./sql_app.db"


engine = create_engine(
    SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False}
)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()

```

سپس یک سشن نیاز داریم که درنهایت بتوانیم به آن سشن وصل شویم . برای این منظور در ابتدا کریت انجین را ایمپورت میکنیم که یک انجین داشته باشیم از اسکیوال الکمی.

فایل main :

```
from fastapi import Depends, FastAPI, HTTPException
from sqlalchemy.orm import Session
from . import crud, schemas, models
from .database import SessionLocal, engine
```

دیپندز برای دیپندنسی اینجکشن و اچ تی تی پی برای مدیریت خطاها.

کلاس سشن از دیتابیس برای تعامل با دیتابیس. کراد و اسکیماز و مدلها در آینده تعریف میشوند.

سشن لوکال و انجین به ترتیب کلاسی برای تعامل با دیتابیس و موتوری از اسکیوال الکمی میباشند.

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

این خط تمام جدولهای موجود در مدل را در دیتابیس میسازد.

```
app = FastAPI()
```

یک اینستنس با نام اپ از کلاس فست ای پی آی میسازیم.

```
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

تابعی که یک دیتا بیس سشن محیا میکند و از کلمه کلیدی ییلد برای ساختن یک موتور جنریتور استفاده میکند. درنهایت هم دیتابیس را میبندد تا از هدررفت منابع جلوگیری شود.

اندپوینتهای دانشجو:

```
@app.get("/students/")
def read_stus(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
    stus = crud.get_stus(db, skip=skip, limit=limit)
    return stus
```

یک اند پوینت برای دانشجو تعریف میشود که اچ تی تی پی ریکویست را با گت دریافت کند. در ادامه لیمیت و اسکیپ برای خوانش دانشجویا بعنوان کویری پارامتر اضافه میشوند. در ادامه یک دیتابیس سشن به اندپوینت تزریق میشود. همچنین تابع کراد صدا میشود تا دانشجویان از دیتابیس گرفته شوند.

```
@app.get("/students/{stu_id}/")
def read_stu(stu_id: int, db: Session = Depends(get_db)):
    db_stu = crud.get_stu(db, id=stu_id)
    if db_stu is None:
        raise HTTPException(status_code=404, detail="Stu not found")
    return db_stu
```

به سراغ خوانش یک دانشجو میرویم. ابتدا یک دیتابیس سشن را به همراه آی دی به اند پوینت تزریق میکنیم. استیودنت آی دی بعنوان پس پارامتر در یو ار ال قرار میگیرد. تابع کراد را صدا میزنیم تا دانشجو را از دیتابیس فراخوانی کند. شرط میکنیم که آیا دانشجو وجود دارد یا خیر. اگر نداشت مدیریت خطا میکنیم با اچ تی تی پی و ارور 404 میدهیم.

ساختن یک دانشجو:

```
@app.post("/RegStu/")
def create_stu(stu: schemas.Stu, db: Session = Depends(get_db)):
    db_stu = crud.get_stu(db, id=stu.STID)
    if db_stu:
        raise HTTPException(status_code=400, detail="Stu already registered")
    return crud.create_stu(db=db, stu=stu)
```

یک اندپوینت تعریف میکنیم که دانشجویان را با متد پست ایجاد کند. در ریکویست بادی دیتاولیدیتور اسکیماس را صدا زده ایم. در ادامه کراد برسی میکند که آیا دانشجو وجود داشته یا خیر اگر وجود دارد ارور مربوطه نمایش داده میشود. در نهایت اگر دانشجو جدید بود او را به دیتا بیس اضافه میکند.

حذف یک دانشجو:

```python
@app.delete("/DelStu/{STID}/")
def delete_stu(STID: int, db: Session = Depends(get_db)):
    db_stu = crud.get_stu(db, id=STID)
    if not db_stu:
        raise HTTPException(status_code=400, detail="Stu not exist")
    return crud.delete_stu(db=db, id=STID)
```

یک اندپوینت برای حذف دانشجو با استیودنت ای دی و به وسیله اچ تی تی پی تعریف میشود.

در ادامه با کراد برسی میکند که آیا دانشجو وجود دارد یا خیر.

آپدیت یک دانشجو:

```python
@app.put("/UpdateStu/{STID}/")
async def update_stu(STID: int, stu: schemas.Stu, db: Session = Depends(get_db)):
    db_stu = crud.get_stu(db, id=STID)
    if not db_stu:
        raise HTTPException(status_code=400, detail="Stu not exist")
    return crud.update_stu(db=db, id=STID, stu=stu)
```

طبق معمول اندپوینت تعریف مشود. با کراد برسی میشود که آیا دانشجو وجود دارد. اگر داشت ارور نمایش داده میشود. اگر همه چیز خوب پیش رفت دانشجو آپدیت میشود.

برای اساتید و دروس نیز طبق همین محتوا به پیش میرویم.

فایل اسکیماز:

ماژولهای ضروری را ایمپورت میکنیم:

```python
from pydantic import BaseModel, Field, validator
```

```python
import os
import json
from .database import SessionLocal
from . import crud
```

او اس یک ماژول برای برقراری ارتباط با سیستم عامل.

جیسون یک ماژول برای کار با داده های جیسونی.

سشنال لوکال از دیتا بیس داخلی خودمان ایمپورت شده تا سشنهای جدیدی برای کار با دیتا بیس بسازد.

لوکال ماژول کراد هم برای ساخت و خواندن و آپدیت و حذف کردن.

تعریف کلاس استاد:

```python
class Professor(BaseModel):
    pk: int
    LID: int
    Fname: str
    Lname: str
    ID: str
    Department: str
    Major: str
    Birth: str
    BornCity: str
    Address: str = Field(max_length=100)
    PostalCode: str = Field(pattern=r"^[0-9]{10}$")
    CPhone: str = Field(pattern=r"^((\+98|0|098)9\d{9})$")
    HPhone: str = Field(pattern=r"^0[1|3|4|5|6|7|8|9][0-9]{9}$|^02[0-9]{9}$")
    Lesson_ids: str
    # LCourseIDs: list[Lesson] = []
```

صحت سنجی

```python
@validator("LID")
def validate_LID(cls, value):
    if len(str(value)) != 6:
        raise ValueError("LID must be 6 digits.")
    return value

@validator("Fname")
def validate_Fname(cls, value):
    if len(value) > 10:
        raise ValueError("first name is too long (must be less than 10 characters)")
    for i in value:
        if i not in persian_char:
            raise ValueError("first name must be only contain persian characters")
    return value
```

این فرایند برای دانشجو و دروس نیز تکرار میشود.

فایل مدلها:

```
from sqlalchemy import Boolean, Column, ForeignKey, Integer, String, Table
from sqlalchemy.orm import relationship
from .database import Base
```

بولین- کالم- فارن کی- اینتیجر- استرینگ و جدول برای تعریف جدولها و ستونها.

ریلیشن شیپ برای تعریف رابطه بین جدولها. کلاس بیس از دیتابیس داخلی برای تعریف اسکیمای دیتابیس.

ستونها:

```
student_lesson_association = Table(
    'student_lesson',
    Base.metadata,
    Column('student_id', Integer, ForeignKey('students.STID')),
    Column('lesson_id', Integer, ForeignKey('lessons.CID'))
)

student_professor_association = Table(
    'student_professor',
    Base.metadata,
    Column('student_id', Integer, ForeignKey('students.STID')),
    Column('professor_id', Integer, ForeignKey('professors.LID'))
)

professor_lesson_association = Table(
    'professor_lesson',
    Base.metadata,
    Column('professor_id', Integer, ForeignKey('professors.LID')),
    Column('lesson_id', Integer, ForeignKey('lessons.CID'))
)
```

تعریف کلاس دانشجو:

```
class Student(Base):
    __tablename__ = "students"
    pk = Column(Integer, primary_key=True, unique=True, index=True)
    STID = Column(Integer, unique=True)
    Fname = Column(String)
    Lname = Column(String)
    Father = Column(String)
    Birth = Column(String)
    IDS = Column(String)
    BornCity = Column(String)
    Address = Column(String)
    PostalCode = Column(String)
    CPhone = Column(String)
    HPhone = Column(String)
    Department = Column(String)
    Major = Column(String)
    Married = Column(Boolean)
    ID = Column(String, unique=True)
    Courses_ids = Column(String)
    Professor_ids = Column(String)
```

به همین صورت برای اساتید و دروس تکرار میشود.

در نهایت ریکوایرمنت های پروژه را یا دستی یا با استفاده از یک پکیج داخلی پایتون به نام پیپ رکس تهیه میکنیم.

در ادامه یک فایل داکر در یک دایرکتوری بالا تر از فایل اپ خود میسازیم.

داکرایز کردن:

```
FROM python:3.12.4

WORKDIR /var/www

COPY /app/requirements.txt .

RUN pip install -r requirements.txt

COPY app .

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

ورژن پایتونی پایتونی که با آن کد زدیم در بالا قرار میگیرد.

ورک دایرکتوری دلخواه است اما ترجیح است که ثابت بماند.

در ادامه فایل ملزومات کپی میشوند.

دستور ران شدن و نصب فایل ملزومات داده میشود.

کل محتوای اپ کپی میشود.

دستورات سی ام دی با اسم کانتینر فست ای پی ای داده میشود. در ادامه پروژه را کلون کرده سپس محتوای پروژه اد و کامیت میشود. در نهایت پوش به گیتهاب.

پایان