فایل گزارش پروژه انتخاب واحد

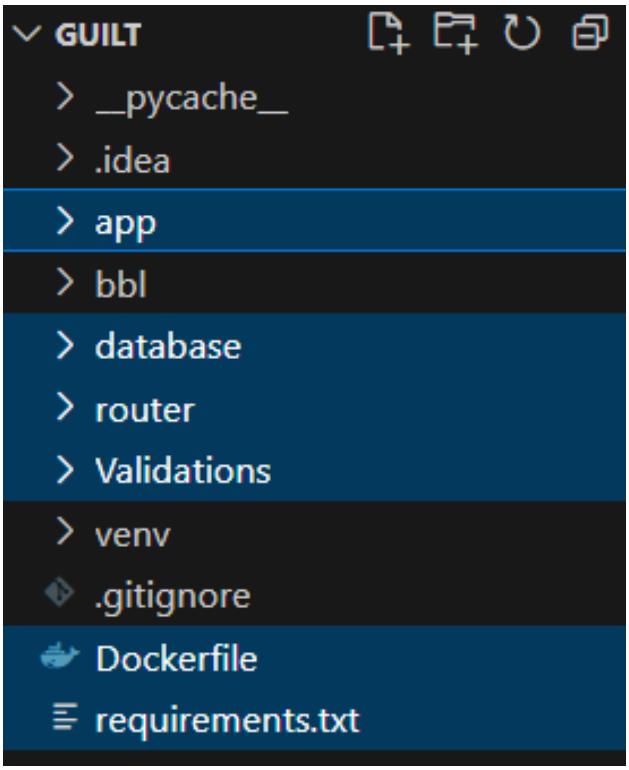
کیان مقدسی

40211415058

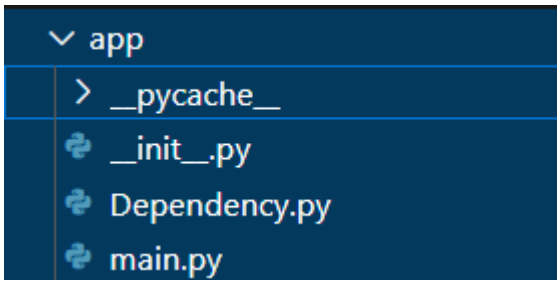
استاد : دکتر رشنو

درس: برنامه نویسی پیشرفته

نگاهی کلی به پروژه:



پکیج app:



ماژول Dependency:

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

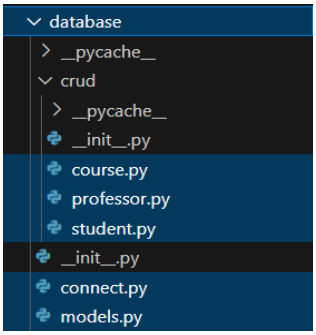
در این تابع ارتباط با database برقرار میکنیم و در کل پروژه برای هرگونه استفاده و ارتباط با database این تابع صدا زده میشود.

ماژول main :

```python
class config:
    orm_mode = True


app = FastAPI()



app.include_router(student.app, tags=["Student"])
app.include_router(professor.app, tags=["Professor"])
app.include_router(course.app, tags=["Course"])
```

در این ماژول تمام API های نوشته شده در پکیج router را گرفته و در این ماژول صدا میزنیم و برای مرتب بودن و تشخیص بهتر API ها آنها را tag گذاری میکنیم.

پکیج database:



پکیج crud:

در این پکیج تمام توابع مختلف که توسط API های ما برای درج , ثبت, آپدیت و حذف نیاز دارد قرار دارد برای مثال:

```
def get_course(db:Session,cid:int):
    return db.query(models.Course).filter(models.Course.CID==cid).first()
```

ای تابع درس مورد نظر را با دریافت کد درس و صدا زدن جدول با این کد درس از جدول داده ها درس مورد نظر را به کاربر نمایش میدهد.

```
def create_course(db:Session,course:schemas.Coursbase):
    db_course=models.Course(CID=course.CID,Cname=course.Cname,Department=course.D
epartment,Credit=course.Credit)
    db.add(db_course)
    db.commit()
    db.refresh(db_course)
    return db_course
```

این تابع با گرفتن تمام اطلاعات از basemodel که اطلاعات توسط یک body request به آن ارسال شده و آنها را دانه به دانه وارد جدول میکند و سپس این اطلاعات را db.add که با db سشن را شروع میکند و add تابع وارد کردن اطلاعات به جدول است و باز سشن و تابع commit تا اطلاعات واقعا وارد جدول شود و refresh که جدول را آپدیت میکند تا اطلاعات وارد شده را به نمایش بگذارد و در آخر اطلاعات وارد شده به کاربر نشان داده میشود و این نشانه بر موفقیت عملیات است.

```
def update_Course(db: Session, Course_id: int, Course_update: schemas.Courseup):
    Course = db.query(models.Course).filter(models.Course.CID ==
Course_id).first()
    if not Course:
        return None

    update_data = Course_update.dict(exclude_unset=True)
    for key, value in update_data.items():
        setattr(Course, key, value)

    db.add(Course)
    db.commit()
    db.refresh(Course)
    return Course
```

این تابع یکی از سطر های جدول را با توجه به اطلاعات ورودی تغییر میدهد که در ابتدا آن خط جدول صدا زده شده سپس عملیات آپدیت بر روی آن اجرا شده است که اطلاعات وارد شده به یک دیکشنری تبدیل میشوند با این کار میتوانیم از هرکدون از اطلاعات وارد شده برای تابع setattr استفاده شود تا بتواند تغییرات انجام شده را وارد کند.

```
def delete_cous(db,cid:int):
    cous = db.query(models.Course).filter(models.Course.CID == cid).first()
    db.delete(cous)
    db.commit()
    return 'درس مورد نظر پاک شد.'
```

این تابع سطر مورد نظر جدول را صدا زده و بر آن تابع delete را که باعث می شود آن سطر از جدول پاک شود اجرا میشود و در آخر عملیات تایید میشود و پیام مدنظر برگردانده میشود.

علاوه بر این دانشجو و استاد توابعی دارند برای اضافه کردن به جدول های ارتباطی میان آنها و درس برای مثال:

```
def add_student_professer_relation(db:Session , student_id:int , prof_id:int):
   try:
    db_stud = db.query(models.Student).filter(models.Student.STID ==
student_id).first()
    db_prof = db.query(models.Prof).filter(models.Prof.LID==prof_id).first()
    db_check=db.query(models.Stu_profs_association).filter((models.Stu_profs_asso
ciation.c.Student_ID == student_id) &
(models.Stu_profs_association.c.Proffesrt_ID == prof_id)).first()
    if not db_stud:
        return False , ".دانشجو وجود ندارد"

    if not db_prof:
        return False , ".استاد وجود ندارد"

    if db_check:
        return False , ".استاد مورد نظر قبلا انتخاب شده است"
    db_stud.LIDs.append(db_prof)
    db.commit()
    return True , ".استاد با موفقیت برای برای دانشجو انتخاب شد"
   except BaseException as Nigg:
       print(Nigg)
       return False , ".استاد مورد نظر قبلا ثبت شده است"
```

این تابع برای اضافه کردن به جدول ارتباطی میان دانشجو و استاد است که در ابتدا خط های مورد نظر از دو جدول استاد و دانشجو و جدول ارتباطی میان دانشجو و استاد صدا زده میشود چک میشود که آیا سطر های مورد نظر برای استاد و دانشجو و جدول ارتباطی میان آن دو وجود دارد یا نه

سپس با انجام تمام تاییدیه ها یا دانستن آنکه جدول ارتباطی میان این دو ساخته شده است

```
db_stud.LIDs.append(db_prof)
```

این خط اطلاعاعت استاد و دانشجو را وارد جدول ارتباطی آنها میکند.

و برای حذف سطر از جدول استاد تنها تفاوت در این بجای تابع append از تابع remove استفاده میشود.

ماژول connect:

```
SQLALCHEMY_DATABASE_URL = "sqlite:///./sql_app.db"
# SQLALCHEMY_DATABASE_URL = "postgresql://user:password@postgresserver/db"

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

Base = declarative_base()
```

این ماژول برای راه اندازی sqlalchemy هست و ارتباط بین database و API را برقرار میکند.

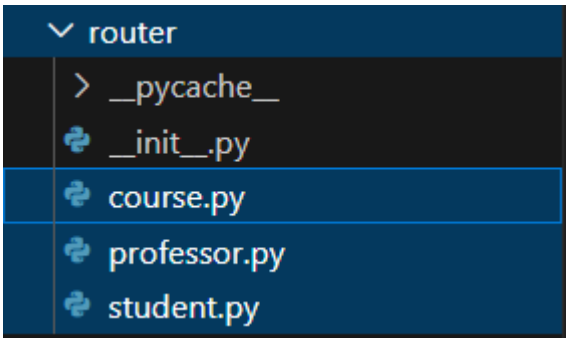
ماژول models:

در این ماژول ساختار جداول ما مشخص میشود برای مثال جدول درس:

```
class Course(Base):
    __tablename__ = "Course"
    CID = Column(Integer, primary_key=True, unique=True)
    Cname = Column(String)
    Department = Column(String)
    Credit = Column(Integer)
```

که یک کلاس است که از فایل کانکت شی Base را به ارث برده بعد آن نام جدول مشخص میشود سپس رکورد های آن مشخص میشود برای مثال CID یکی از رکود هاست که عدد فقط میگیرد و کل جدول توسط آن صدا زده میشود و خاص است .

پکیج router:



این پکیج تمام API های مارو دارد و اطلاعات را از کاربر گرفته و ماژول ها وتوابع داخل ماژول های پکیج crud را برای اجرای عملیات های مد نظر صدا میکند.

که ماژول course عملیات های جدول درس , ماژول professor عملیات های جدول اساتید , و ماژول student عملیات های جدول دانشجو را صدا میزند. برای مثال:

```
@app.post("/regcous/", response_model=schemas.Coursbase)
def create_cours(cours: schemas.Coursbase, db: Session = Depends(get_db)) ->
schemas.Coursbase:
    validator = ValdationsC.CousVald
    validate_data = validator.validate(cours.dict())
    if any(value for value in validate_data.values()):
        raise HTTPException(status_code=400, detail=validate_data)
```

```
    if int(len(str(cours.CID))) > 5:
        raise HTTPException(status_code=406,detail="CID invalid")


    db_course = course.get_course(db, cid=cours.CID)
    if db_course:
        raise HTTPException(status_code=400, detail="CID already registerd.")
    return course.create_course(db=db, course=cours)
```

این API برای وارد کردن یک درس به جدول درس است اطلاعات را با متود post که اطلاعات را توسط یک body request گرفته .

در ابتدا این فایل validation درس را در یک شی میگذاریم سپس اطلاعات گرفته شده را تبدیل به یک دیکشنری میکنیم تا بتوانیم از آنها استفاده کنیم و به فایل validation میفرستیم سپس یک شرط میگذاریم و توسط آن اگر جوابی از طرف فایل آمد یعنی ورودی استاندارد نبوده و یک ارورر میخوانیم و به کاربر نشان میدهیم تا اطلاعات ورودی را تصحیح کند و اگر جوابی از طرف فایل نیامد این شرط رد میشود سپس تابع ساخت جدول صدا زده شد از crud و اطلاعات وارد جدول میشوند.

مثال :

```
@app.patch("/upcor/{Course_id}", response_model=schemas.Courseup)
def update_Course(Course_id: int, Course_update: schemas.Courseup, db: Session =
Depends(get_db)):
    validator = ValdationsC.CousVald
    validate_data = validator.validate(Course_update.dict())
    if any(value for value in validate_data.values()):
        raise HTTPException(status_code=400, detail=validate_data)


    Course = course.get_course(db=db, cid = Course_id)
    if Course is None:
        raise HTTPException(status_code=404, detail="Course not found")

    updated_Course = course.update_Course(db, Course_id= Course_id,
Course_update=Course_update)
```

```
    if updated_Course is None:
        raise HTTPException(status_code=404, detail="Course not found")

    return updated_Course
```

برای آپدیت متود patch استفاده میشود و تمام مراحل Ivalidation انجام شده تابع آپدیت از crud صدا زده میشود و اطلاعات سطر مورد نظر آپدیت شده و به کار برگردانده میشود.

```
@app.get("/delcor/{CID}")
def delete_cours(CID:int ,db:Session=Depends(get_db)):
    db_course = course.get_course(db,cid=CID)
    if  db_course is None:
        raise HTTPException(status_code=400,detail="CID Dose not exist.")
    return course.delete_cous(db=db, cid=CID)
```

کد درس با متود get بصورت pathparameter از کاربر گرفته میشود.

برای حذف سطر در جدول درس سطر مورد نظر صدا زده میشود و تایید میشود که وجود دارد پس از تایید وجود تابع حذف از crud صدا زده میشود و سطر حذف میشود.

```
@app.put("/addprfcor/{LID}/{CID}")
def procous(LID:int,CID:int,db:Session=Depends(get_db)):
    success, message = professor.add_Proffeser_course_relation(db=db, LID = LID,
CID=CID)
    if success:
        return {"message": message}
    else:
        raise HTTPException(status_code=400, detail=message)
```

این API برای استاد درس را وارد میکند که تابع اضافه کردن اطلاعات به جدول ارتباطی درس و استاد را صدا میکند اطلاعات مورد نیاز را در آن قرار

میدهد و در جواب دو شی را از آن میگیرد که اگر شی اول موجود بود یعنی اضافه شده و شی دوم را برمگیرداند و اگر موجود نبود یعنی مشکلی پیش آمده و ارورر ایجاد کرده و شی دوم را بر میگرداند.

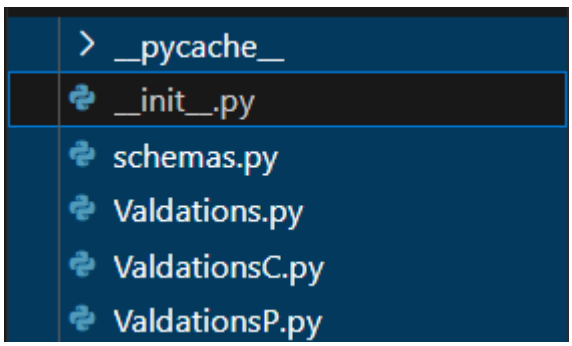
برای حذف درس برای استاد نیز همین گونه صدا زده میشود.

تنها نکته باقی مانده این است که برای دانشجو و استاد در زمان نشان دادنشان در تابع get آنها در crud یک joinload رخ میدهد تا برای مثال برای دانشجو درسها و استاد های دانشجو که بعدا اضافه شده اند به کاربر نشان داده بشود بصورت زیر:

```
async def get_student(db:Session,stu:int):
    try:
       query=db.query(models.Student).filter(models.Student.STID==stu).options(jo
inedload(models.Student.ScourseIDs),joinedload(models.Student.LIDs)).first()
       return (True, query , [])
    except BaseException as nig:
        print(nig)
        return (False , {} , ["student dosent exixst."])
```

تمام توابع و API های جداول درس و استاد و دانشجو مانند مثال های بالا نوشته شده است .

پکیج Validations:



ماژول schema:

این ماژول تمام نحوه های ورودی را برای متود post را ساخته و توسط API ها استفاده میشود و مشخص میکند که ورودی چه نوعی باشند(رشته , عدد و.....).

ماژول های Validations وValidationp وValidationc:

به ترتیب تایید اطلاعات دانشجو و استاد و درس است که در API های وارد کردن و آپدیت کردن استفاده میشود تا مطمئن شویم اطلاعات واردی طبق استاندارد گفته شده است که همه این ساختار را دنبال میکنند:

```
class StudentVald(schemas.Stubase):
    resp: ClassVar[Dict[Any, Any]] = {}


    @classmethod
    def validate(cls,values: Dict[Any,Any]):
        cls.resp.clear()
```

برای این مثال از ماژول دانشجو استفاده کرده ایم کلاس ساخته شده از کلاس دانشجو داخل ماژول schema اطلاعات را گرفته سپس یک شی بنام resp میسازم که یک دیکشنری است تا جواب توابع تایید اطلاعات را در آن قرار دهیم

```
        snum1(values.get('STID'),cls.resp)
        fname(values.get('Fname'),cls.resp)
        lname(values.get('Lname'),cls.resp)
        father_name(values.get('Father'),cls.resp)
        date_sham(values.get('Birth'),cls.resp)
        passy(values.get('IDS'),cls.resp)
        prov(values.get('Borncity'),cls.resp)
        addres(values.get('Address'),cls.resp)
```

```
        postadd(values.get('PostalCode'),cls.resp)
        pnumb(values.get('Cphone'),cls.resp)
        hnumb(values.get('Hphone'),cls.resp)
        coll(values.get('Department'),cls.resp)
        fos(values.get('Major'),cls.resp)
        mrstate(values.get('Married'),cls.resp)
        codemel(values.get('ID'),cls.resp)



        return cls.resp
```

سپس در آخر تمام توابع تایید اطلاعات را صدا زده اطلاعاتی که میخواهیم تایید بشند را که از schema گرفته ایم قرار داده و جواب بر میگرداند و در آخر کل دیکشنری را برگردانده تا در API استفاده شود.

فایل های Dockerfile و requirement.txt:

Dockerfile:

```
FROM python:3.12

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

ENV PYTHONPATH=/src

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

Requirement.txt:

```
fastapi==0.111.0
pydantic==2.8.2
SQLAlchemy==2.0.30
```

در فایل داکر ورژن پایتون در خط اول مسیری اجرایی کامند های زیر خط دوم و سوم فایل requirement **را استفاده میکند و** CMD **دستوری که موقع اجرا کانتینر زده میشود .**

در requirement.txt **نیز ورژن کتابخانه هایی که استفاده بیان میشود.**