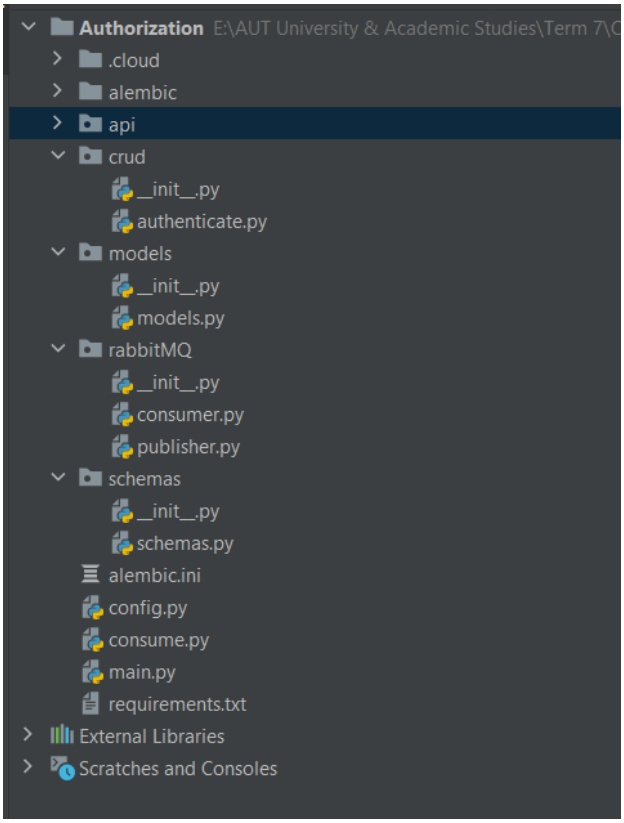
علی بازشوشتری ۹۹۲۳۰۰۸

گزارش تمرین اول درس رایانش ابری

Apiهای این تمرین به کمک FastAPI زده شده اند. ساختار کلی پروژه به شرح زیر است



فایل config.py جهت انجام یک سری تنظیمات و اتصالات نظیر دیتابیس و RabbitMQ می باشد.

در فایل main.py، apiهای سرویس اول نوشته شده اند.

در فایل models.py، ساختار جدول دیتابیس مشخص شده است.

در فایل schemas.py ساختار اشیایی که پاس داده می شوند مشخص شده است.

همچنین در این پروژه از SQLAlchemy ORM استفاده شده است.

فایل publisher.py جهت قرار دادن درخواست های کاربر در صف RabbitMQ است.

فایل های consume.py و consumer.py جهت برداشتن درخواست کاربر از صف و همچنین پردازش آن است که این پردازش شامل دریافت عکس های کاربر از ذخیره ساز S3 و ارسال آنها به IMAGGA و در نهایت ارسال نتیجه درخواست از طریق ایمیل به کمک mailgun می باشد.

```python
SQLALCHEMY_DATABASE_URL = 'postgresql://ABazshoushtari:6X1zAPnsrOto@ep-still-violet-83723632.us-east-2.aws.neon.tech/neondb'
RABBITMQ_URL = "amqp://ybesyakb:NTx4HnKNAh7VLfCwV9nkSls-UKccq-b8@woodpecker.rmq.cloudamqp.com:5672/ybesyakb"

engine = create_engine(
    SQLALCHEMY_DATABASE_URL,
    pool_pre_ping=True,
    pool_recycle=900,
    pool_size=20,
    max_overflow=50
)

SessionFactory: Session = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()


def get_db() -> Session:
    db = scoped_session(SessionFactory)()
    try:
        yield db
    finally:
        db.close()


def get_rabbitMQ_connection():
    params = pika.URLParameters(RABBITMQ_URL)
    connection = pika.BlockingConnection(params)
    return connection
```

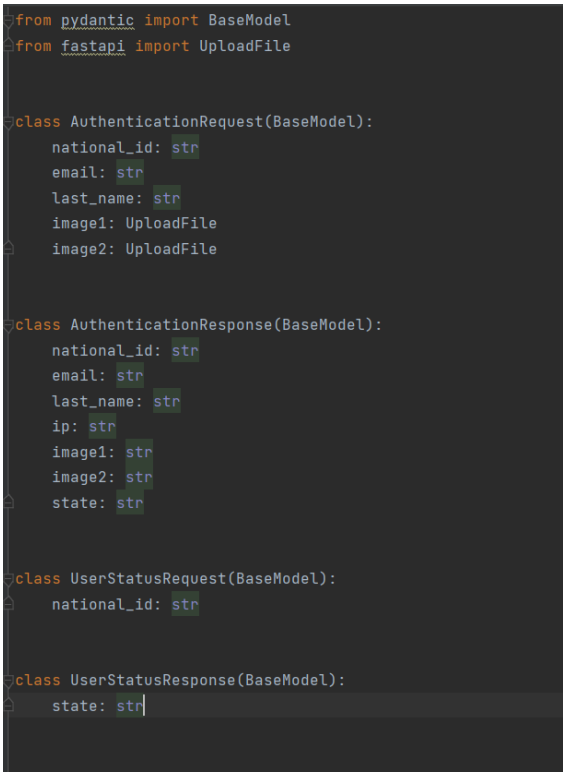
فایل config.py

```python
from sqlalchemy import Column, Integer, String

from config import Base


class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    national_id = Column(String, unique=True)
    email = Column(String, index=True)
    last_name = Column(String)
    ip = Column(String)
    image1 = Column(String)
    image2 = Column(String)
    state = Column(String, default="pending")
```

فایل models.py جهت تعیین شِما و ساختار دیتابیس



```python
from pydantic import BaseModel
from fastapi import UploadFile


class AuthenticationRequest(BaseModel):
    national_id: str
    email: str
    last_name: str
    image1: UploadFile
    image2: UploadFile


class AuthenticationResponse(BaseModel):
    national_id: str
    email: str
    last_name: str
    ip: str
    image1: str
    image2: str
    state: str


class UserStatusRequest(BaseModel):
    national_id: str


class UserStatusResponse(BaseModel):
    state: str
```

فایل shemas.py که اشیای مورد استفاده در پروژه که پاس شده اند را نشان میدهند. این اشیا و کلاس ها از کلاس BaseModel ارث بری می کنند.

```python
app = FastAPI()


@app.post("/users")
def authenticate_user(
        client_request: Request,
        image1: UploadFile = File(...),
        image2: UploadFile = File(...),
        db: Session = Depends(get_db),
        national_id: str = Form(...),
        email: str = Form(...),
        last_name: str = Form(...)

):
    client_ip = client_request.client.host
    return crud.authenticate.create_user_formfile(image1, image2, client_ip, db, national_id, email, last_name)
    # return crud.authenticate.create_user(db=db, user_in=form_file, client_ip=client_ip)
```

Api مربوط به ثبت درخواست در سرویس اول

در این API که یک متود post است، کاربر اطلاعات لازم را برای ما ارسال می کند. فیلد client_request برای گرفتن آیپی کاربر از هدر بسته ها می باشد. فیلد db نیز برای اتصال به دیتابیس و افزودن اطلاعات به دیتابیس است.

در ادامه فانکشن create_user_formfile از فایل authenticate در پکیج crud را مشاهده میکنیم.

```
def create_user_formfile(
        image1: UploadFile,
        image2: UploadFile,
        client_ip: str,
        db: Session,
        national_id: str = Form(...),
        email: str = Form(...),
        last_name: str = Form(...)
):
    user: User = User(
        national_id=encode_base64(national_id),
        email=email,
        last_name=last_name,
        ip=client_ip,
    )

    try:
        db.add(user)
        db.commit()
        img1_format = image1.filename.split(".")[-1]
        img2_format = image2.filename.split(".")[-1]
        user.image1 = str(user.id) + "_1" + f".{img1_format}"
        user.image2 = str(user.id) + "_2" + f".{img2_format}"
        image1.filename = user.image1
        image2.filename = user.image2
        db.commit()
        publish_username(username=f"{user.id}")
    except Exception:
        db.rollback()
        raise
    add_to_s3(image1)
    add_to_s3(image2)
    return user
```

فرمت ذخیره سازی نام عکس های کاربر به صورت روبرو است: userID_1 و userID_2 به همراه پسوند عکس هایی که کاربر آپلود کرده است.

با استفاده از اطلاعات ارسالی کاربر، یک شی از نوع User ساخته می شود و در دیتابیس درج میشود.

کد ملی فرد نیز به صورت base64 رمزنگاری شده و در دیتابیس ذخیره می شود.

فانکشن publish_username از فایل publisher.py وظیفه دارد تا userID کاربر را در صف RabbitMQ قرار دهد تا سرویس دوم آن را از روی صف بخواند و به آن رسیدگی کند.

```python
from config import get_rabbitMQ_connection


def publish_username(username: str):
    connection = get_rabbitMQ_connection()

    channel = connection.channel()
    channel.queue_declare(queue='username')
    channel.basic_publish(
        exchange='',
        routing_key='username',
        body=username
    )
    connection.close()
```

همانطور که مشخص است، کانکشن RabbitMQ از فایل config دریافت می شود.

پس از آن نیز عکس های ارسالی کاربر با همان فرمت ذکر شده، در ذخیره ساز S3 نیز ذخیره می شود.

```python
def add_to_s3(image: UploadFile):
    try:
        s3_resource = boto3.resource(
            's3',
            endpoint_url='https://hw1-cloudcomputing-ali.s3.ir-thr-at1.arvanstorage.ir',
            aws_access_key_id='b2a0b10c-5e30-4735-9d3e-f8bf299542b4',
            aws_secret_access_key='0ecb05b5d60b164f0d44891912e19b0accb253e07f588c73b903bbdd4dc9108d'
        )
    except Exception as exc:
        logging.error(exc)
    else:
        try:
            bucket = s3_resource.Bucket('hw1-cloudcomputing-ali')
            object_name = image.filename
            bucket.put_object(
                ACL='public-read',
                Body=image.file,
                Key=object_name
            )
        except ClientError as e:
            logging.error(e)
```

فانکشن add_to_s3 که عکس های کاربر را در ذخیره ساز s3 ذخیره می کند.

```python
@app.get("/users/status")
def authenticate_user(
        db: Session = Depends(get_db),
        *,
        request,
        client_request: Request
):
    user = crud.authenticate.get_user_status(db=db, national_id=request)
    status = user.state
    user_id = user.id
    ip = client_request.client.host
    if ip != user.ip:
        return {"message:": "Invalid access"}
    if status == "pending":
        return {"message:": "Your authorization request is processing"}
    elif status == "rejected":
        return {"message:": "Your authorization request is rejected"}
    elif status == "accepted":
        return {"message:": f"Your authorization request with id {user_id} is accepted successfully"}
```

API مربوط به بررسی وضعیت درخواست از سرویس اول

این API نیز یک متود GET را پیاده سازی می کند که در آن کلاینت کد ملی خود را ارسال می کند و سپس با توجه به آن، ردیف مربوطه از دیتابیس خوانده می شود و یک شی از نوع User برگردانده می شود. از روی این شی، فیلدهای وضعیت و آیپی را دریافت می کنیم. در صورتی که آیپی درخواست زده شده، با آیپی ای که قبلا در دیتابیس ثبت شده تطابق نداشت، پیغام دسترسی غیرمجاز چاپ می شود؛ در غیر اینصورت نیز با توجه به وضعیت درخواست کاربر، پیغام مناسب چاپ می شود.

فانکشن get_user_status نیز به صورت زیر است.

```python
def get_user_status(
        db: Session,
        national_id,
) -> User:  #schemas.schemas.UserStatusResponse:
    user: User = db.query(User).filter(User.national_id == encode_base64(national_id)).first()

    # return schemas.schemas.UserStatusResponse(state=user.state)
    return user
```

اکنون به توصیف و توضیح سرویس دوم می پردازیم؛ جایی که درخواست های کاربر از صف برداشته شده و پردازش و رسیدگی می شوند. این سرویس توسط فایل consume.py اجرا می شود ک در آن فانکشن consume_username از فایل consumer.py فراخوانی می شود.

consumer.py consume.py

```python
from rabbitMQ import consumer

consumer.consume_username()

```

در فایل consumer.py، یک متغیر گلوبال به نام db جهت دسترسی به دیتابیس تعریف میکنیم. که فانکشن next را بر روی get_db() از فایل config.py فراخوانی میکند و نتیجه را در متغیر db ذخیره می کند.
Next جهت برگرداندن آیتم بعدی از iterator می باشد و در اینجا عملا ما یک کانکشن و Session به دیتابیس را میگیریم.

consumer.py consume.py

```python
from config import get_rabbitMQ_connection, get_db
from sqlalchemy.orm import Session
from sqlalchemy import select
from fastapi import Depends
import requests
from config import get_db
from models.models import User

db = next(get_db())

```

```python
def consume_username():
    while True:
        try:
            connection = get_rabbitMQ_connection()
            channel = connection.channel()
            channel.queue_declare(queue='username')

            channel.basic_consume('username', callback, auto_ack=True)
            print(' [*] Waiting for messages:')
            channel.start_consuming()

            connection.close()
        except Exception:
            print("Consumer failed. restarting.")
```

مطابق کد بالا یک اتصال به صف RabbitMQ برقرار شده و شروع به consume کرد از آن میکنیم. این consume کردن همراه با فراخوانی متود callback می باشد.

```python
def callback(ch, method, properties, body):
    print(" [x] Received " + str(body))
    # parse str(body) and get id
    user_id = int(str(body).split("'")[1])
    user = db.query(User).filter(User.id == user_id).first()
    user.state = "accepted"  # will change later if auth fails
    # get img1 and img2 fields from db
    user_img1_filename = user.image1
    user_img2_filename = user.image2
    # dont download from S3, give below url to IMGGA:

    image1_url = 'https://hw1-cloudcomputing-ali.s3.ir-thr-
at1.arvanstorage.ir/hw1-cloudcomputing-ali/' + user_img1_filename
    image2_url = 'https://hw1-cloudcomputing-ali.s3.ir-thr-
at1.arvanstorage.ir/hw1-cloudcomputing-ali/' + user_img2_filename

    print(image1_url)
    print(image2_url)

    face_id_1 = face_detection(image1_url)
    face_id_2 = face_detection(image2_url)

    if not (face_id_1 and face_id_2):
        user.state = "rejected"
    elif not face_similarity(face_id_1, face_id_2):
        user.state = "rejected"

    # update state in db
    db.commit()

    # send email
    send_email(user.email, user.state)

    print(" [@] Proccessed User request with user id : " + str(user.id))
```

user_id کاربر از روی صف خوانده می شود و به کمک آن، ردیف مربوطه از دیتابیس خوانده و دریافت می شود. وضعیت درخواست نیز به طور موقت به accepted تغییر پیدا می کند. (دقت شود که تغییری در دیتابیس در این مرحله رخ نمی دهد.)

همچنین URL محل ذخیره سازی عکس ها نیز با توجه به آنچه از دیتابیس دریافت شده، تولید می شود تا در ادامه برای API Callهای مربوط به تشخیص و شباهت چهره استفاده شود.

در ادامه فانکشن face_detection فراخوانی می شود و در آن یک API Call انجام می شود تا متوجه شویم آیا در عکس های ارسالی کاربر چهره ای وجود داشته است یا خیر.

```
def face_detection(image_url: str):
    api_key = 'acc_912ceb0481c3d1c'
    api_secret = '48d3db05564d3ffda3aa5ce812f2f9e7'

    response = requests.get(
        'https://api.imagga.com/v2/faces/detections?image_url=' + image_url + "&return_face_id=1",
        auth=(api_key, api_secret))
    print(response.json())
    if len(response.json()["result"]["faces"]) < 1:
        return None
    face_id = response.json()["result"]["faces"][0]["face_id"]
    return face_id
```

در شرط چک می شود که اگر پاسخ دریافتی از IMAGGA دارای face_id نبود پس یعنی چهره ای تشخیص داده نشده و None برگردانده میشود. در غیر این صورت نیز face_id برگردانده می شود.

اگر هر یک از عکس ها چهره نداشته باشند، وضعیت کاربر به صورت rejected تغییر پیدا میکند.

در ادامه اگر هر دو عکس ارسالی دارای چهره بودند، فانکشن face_similarity فراخوانده می شود و در آن API Call مربوط به شباهت چهره انجام می شود.

```
def face_similarity(first_face_id: str, second_face_id: str):
    api_key = 'acc_912ceb0481c3d1c'
    api_secret = '48d3db05564d3ffda3aa5ce812f2f9e7'

    response = requests.get(
        'https://api.imagga.com/v2/faces/similarity?face_id=%s&second_face_id=%s' % (first_face_id, second_face_id),
        auth=(api_key, api_secret))
    if float(response.json()["result"]["score"]) < 80:
        return False
    return True
```

در این تابع face_idها برای IMAGGA از طریق متود get به صورت کوئری پارامتر ارسال می شوند. پس از آن نیز در پاسخ دریافتی اگر درصد شباهت کمتر از ۸۰ درصد بود، False و در غیر اینصورت True برگردانده می شود.

پس از آن نیز دیتابیس آپدیت می شود.

در نهایت نیز وضعیت درخواست کاربر به کمک فانکشن send_email از طریق ایمیل برای او ارسال می گردد.

```
def send_email(email: str, state: str):
    if state == "accepted":
        authorization_report = "Your Authorization request accepted successfully."
    else:
        authorization_report = "Your Authorization request rejected."
    return requests.post(
        "https://api.mailgun.net/v3/sandbox90f355c2726b4e8789a7ec98994be7d9.mailgun.org/messages",
        auth=("api", "914cde4d0ed00b5761357e7b61b5a52c-3e508ae1-38d1bc09"),
        data={"from": "Excited User <mailgun@sandbox90f355c2726b4e8789a7ec98994be7d9.mailgun.org>",
              "to": [email],
              "subject": "Authorization State",
              "text": authorization_report})
```

این تابع ایمیل کاربر و وضعیت را به صورت ورودی دریافت می کند و پیغام مناسب را برای او ارسال می کند.

سرویس های ابری مورد استفاده در این تمرین:

DBaaS: سرویس postgres بر روی پلتفرم neon.tech

S3: ابرآروان

RabbitMQ: CloudAMQP

سرویس پردازش تصویر: IMAGGA

سرویس ارسال ایمیل: mailgun

میزبان ابری: رانفلر