گزارش پروژه اول رایانش ابری – آشنایی با برخی خدمات ابری

سرویس اول

API مربوط به آپلود فایل:

برای پیاده سازی این بخش ما نیاز به یک سرور http (که با استفاده از زبان پایتون و فریم ورک fastapi نوشته شد) و دسترسی به پایگاه داده (لیارا) و سرویس s۳ (پارس پک) داریم. برای مدیریت این شرایط فایل main در نظر گرفته شد که به دو فایل مجزای مرتبط با پایگاه داده و سرویس ابری دسترسی دارد و سرور http هم در آن پیاده سازی شده است.

```
@app.post("/submit_email/")
async def submit_email(id: int, email: str, inputs: str, language: str,
enable: int, file: UploadFile = File(...)):
    #insert to db
    query = uploads_table.insert().values(id=id,
                                          email=email,
                                          inputs=inputs,
                                          language=language,
                                          enable=enable)

    await database.execute(query=query)
    address = str(id) + "." + file.filename.split(".")[-١]
    # save file on s٣
    upload_file(file, address)

    return f"Your submission was registered with ID: {id}"
```

در این بخش مشاهده میکنیم که با یک ریکوئست post از طرف کاربر و ارسال موارد خواسته شده از طرف او این اطلاعات در پایگاه داده ذخیره میشود و فایل ارسالی هم در سرویس ابری ذخیره میشود:

برای ذخیره سازی در پایگاه داده لیارا جدول uploads_table تعریف شده که بصورت زیر تعریف شده:

```
uploads_table = sqlalchemy.Table(
    "uploads",
    metadata,
    sqlalchemy.Column("id", sqlalchemy.Integer, primary_key=True),
    sqlalchemy.Column("email", sqlalchemy.String),
    sqlalchemy.Column("inputs", sqlalchemy.String),
    sqlalchemy.Column("language", sqlalchemy.String),
    sqlalchemy.Column("enable", sqlalchemy.Integer)
)


metadata.create_all(engine)
```

همچنین شمای این جدول هم در فایل مجزایی به عنوان شما تعریف شده است:

```
class Upload(BaseModel):
    email: str
    inputs: str
    language: str
    enable: int
```

برای ذخیره سازی در سرویس ابری پارس پک فایلی مجزا برای سرویس s۳ تعریف شده است که تابع upload_file ذخیره سازی را انجام داده و درصورت موفقیت آمیز بودن آن پیام موفقیت را برمیگرداند:

```
def upload_file(file, object_name):
    contents = file.file.read()
    bucket.put_object(
        ACL='private',
        Body=contents,
        Key=object_name
    )
    print(f'INFO:     File-{object_name} uploaded successfully')
```

API مربوط به اجرای فایل:

برای پیاده سازی این بخش در سرور ریکوئست دیگری در نظر گرفته شد که کاربر با ارسال شناسه فایل مورد نظر با استفاده از متد get درخواستش را میفرستد سپس اگر این شناسه در پایگاه موجود نبود ارور میخورد درغیر این صورت چک میکند اگر enable این فایل برابر با ۰ بود به سرویس دوم فرستاده میشود و روندش را آنجا طی میکند در غیر اینصورت پیامی مبنی بر عدم امکان اجرای درخواست کاربر فرستاده میشود:

```
@app.get("/check_email/")
async def check_email(id: int):
    query = uploads_table.select().where(uploads_table.c.id == id)
    result = await database.fetch_one(query=query)
    if not result:
        raise HTTPException(status_code=٤٠٤, detail="Email not found")
    elif result["enable"] == ٠:
        send(id)
    else:
        raise HTTPException(status_code=٤٠٠, detail="You cannot request this
code")

    return {"result": 'Sending to job_table..'}
```

API مربوط به دریافت وضعیت کارها:

در این بخش با دریافت ایمیل کاربر با ریکوئست get آن را در جدول uploads جستجو کرده و تمامی سطر هایی که این ایمیل را دارند را برمیگردانیم سپس اطلاعات لازمه برای نمایش به کاربر را از جدول job_table و results_table , می اوریم و به صورت یک لیست از json ها به کاربر برمیگردانیم:

```
# curl -X GET
"http://localhost:۸۰۰۰/check_user/?email=uni.mahdipour@gmail.com"
@app.get("/check_user/")
async def check_user(email: str):
    print('Got it!')
    result_list = []
    json_objs = await get_requests_by_email(email)
    for job_obj in json_objs:
        print(type(job_obj))
        job = json.loads(job_obj)
        print(f'job : {job}')
        fromJobs = json.loads(get_data_from_job_table(job['id']))
        status = fromJobs['status']
        jobQ = fromJobs['job']
        print(f'status = {status} jobQ = {jobQ}')
        fromResults = json.loads(get_data_from_results_table(job['id']))
        output = fromResults['output']
        filelink = fromResults['filelink']
        execute_date = fromResults['execute_date']
        print(f'output = {output} filelink = {filelink} execute_date =
{execute_date}')
        result_obj = create_json(job['id'], status, jobQ, output,
execute_date, filelink)
        print(f'robj = {result_obj} added to list')
        result_list.append(result_obj)
    print(result_list)
    return {"result": str(result_list)}
```

سرویس دوم

API سوم از سرویس اول درصورت مطابقت درخواست کاربر با شرایط خواسته شده یک شناسا یکتا را به سرویس RabbitMQ میفرستد و این شناسه در صف emails قرار میگیرد، حالا سرویس دوم باید این شناسه هارا از این صف بخواند، سطر مربوط به آن را از جدول uploads_table از پایگاه داده استخراج کند برای این کار ابتدا شرایطی ایجاد میکنیم که متداوما از صف خوانده شود:

```
def main():
    connection = pika.BlockingConnection(pika.URLParameters(AMQP_URL))
    channel = connection.channel()

    channel.queue_declare(queue='emails')

    channel.basic_consume(queue='emails', on_message_callback=callback,
auto_ack=True)

    print(' [*] Waiting for messages. To exit press CTRL+C')
    channel.start_consuming()
```

با دریافت هر شناسه در صف بصورت خودکار تابع زیر فراخوانی میشود:

```
def callback(ch, method, properties, body):
    print(" Emails Received %r" % body)
    gotten_id = str(body).split(".")[0].split("'")[1]
    data = json.loads(get_data_from_db(gotten_id))
    if data["enable"] == 0:
        print('sending to string creator..')
        stringCreator(data)
```

اگر امکان اجرای این درخواست وجود داشت پس از دریافت اطلاعات مربوط به شناسه از جدول قبلی تابعی فراخوانی میشود که مسئولیت ثبت این درخواست در جدول کارها را دارد:

```
def stringCreator(data):
    filename = find_file(data["id"])
    if filename is not None:
        print(f"File Name : {filename}")
        fileData = get_file_content(filename)
        if fileData is not None:
            print(f'File Data: {fileData}')
            queryString = create_json(data['language'], data['inputs'],
fileData)
            # insert to db jobs_table
            query = job_table.insert().values(upload=data['id'],
job=queryString)
            with engine.connect() as conn:
                conn.execute(query)
            print('Added to job_table successfully')
        else:
            print('File Content is Empty')
    else:
        print(f"File with id {data['id']} not found!")
```

همانطور که مشاهده میشود در این بخش اطلاعات لازم برای ثبت در جدول کارها را مرتب کرده، محتوای فایل وجودی و زبان برنامه نویسی را در قالب json در اورده و به رشته تبدیل و به عنوان query string در می اوریم و در نهایت این اطلاعات را در جدول کارها ذخیره میکنم که بصورت زیر تعریف شده است:

```
job_table = sqlalchemy.Table(
    "job",
    metadata,
    sqlalchemy.Column("id", sqlalchemy.Integer, primary_key=True,
autoincrement=True),
    sqlalchemy.Column("upload", sqlalchemy.Integer,
sqlalchemy.ForeignKey("uploads.id")),
    sqlalchemy.Column("job", sqlalchemy.String),
    sqlalchemy.Column("status", sqlalchemy.String, default="none-executed")
)
```

سرویس سوم

در این بخش بصورت بازه های ۱ دقیقه ای جدول کارها را چک میکنیم و کارهایی که وضعیت -none executed دارند را با ایدی اپلودشان بصورت تاپل در یک لیست ذخیره میکنیم که از اضافه شدن دوباره یک کار حین انجام آن جلوگیری شود:

```
async def main():
    job_list = []
    while True:
        new_jobs = await get_new_jobs()
        if new_jobs:
            for job in new_jobs:
                job_obj = json.loads(job)
                job_id = str(job_obj['id'])
                job_dict = {"id": job_id, "job": job_obj}
                if job_dict not in job_list:
                    job_list.append(job_dict)
                    print(f"Added job with ID {job_id} to the list")
            preRunner(job_list)
        else:
            print("No new jobs found, waiting...")
        await asyncio.sleep(١٠)  # Wait for ١٠ seconds before checking again
```

در قدم بعدی کارها توسط تابع preRunner یکی یکی بررسی شده اطلاعات آنها از جدول uploads_table استخراج شده و به codex یک http ریکوئست زده میشود که آن را اجرا کند اگر این اجرا موفق بود ایدی آن و پاسخ codex با وضعیت ۱ که یعنی موفق بوده برای تابع sendMail فرستاده میشود و اگر موفق نبود همین اطلاعات با کد ۰ ارسال خواهد شد که یعنی با خطا روبرو شده ایم:

```
def preRunner(job_list):
    for job in job_list:
        obj = job.get('job')
        code_obj = obj.get('job')
        print(f'obj : {obj}')
        code_data = json.loads(code_obj)
        language = checkLang(code_data['language'])
        url = "https://api.codex.jaagrav.in"
        payload = {
            "code": code_data['contents'],
            "language": language,
            "inputs": code_data['inputs']
        }
        headers = {"Authorization": "Bearer <access_token>"}
        response = requests.post(url, json=payload, headers=headers)
        if response.status_code == ٢٠٠:
            result = response.json()
            print(f'Response : {result}')
            if result['status'] == ٢٠٠:
                sendMail(obj['upload'], ١, result)
            else:
                sendMail(obj['upload'], ٠, result)
```

```
        else:
            print(f"Error running job {job['id']}: {response.text}")
```

در تابع sendMail بنابر وضعیت ورودی اگر موفق بود ایمیلی متناسب با این شرایط به کاربر ارسال میکنیم و وضعیت کار را در جدول کارها به executed تغییر میدهیم و اگر وضعیت ورودی خطارا نشان میداد ایمیلی مبتنی بر این وضعیت برای کاربر ارسال کرده و enable مربوط به درخواست کاربر را در جدول uploads به یک تغییر میدهیم که در ادامه از اجرای آن جلوگیری شود.

```
def sendMail(id, status_id, result):
    uploads_data = json.loads(get_data_from_uploads_table(id))
    email = uploads_data['email']

    if status_id == ٠:
        subject = f'Error while compiling code - {id}'
        status = 'Error'
        enable_off(id)

    elif status_id == ١:
        subject = f'Successful Code result - {id}'
        status = 'Success'
        status_executed(id)

    table_data = [['Response from codeX:', ''], [f'Status: {status}', '']]
    for key, value in result.items():
        table_data.append([f'{key}:', value])

    table = tabulate(table_data, tablefmt="plain")
    text = f'Hi,\n\nYour code request
{status.lower()}ed!\n\n{table}\n\nRegards,\n\nPrecioux'
    send_simple_message(email, subject, text)
    print('Email sent successfully!')
```