دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

تمرین شماره ۱

رایانش ابری

شایان بالی

۹۸۳۱۰۱۴

استاد درس

دکتر جوادی

پاییز ۱۴۰۱

در این تمرین ما به طراحی یک سیستم آگهی خودروی ابری پرداختیم که از دو سرویس تشکیل شده بود که سرویس اول از ۲ API تشکیل شده بود. از آنجایی که به API احتیاج داشتیم من از جنگو استفاده کردم چون امکانات خوبی را جهت طراحی API در اختیار میگذارد. در ادامه به بیان مراحل انجام این تمرین می‌پردازم.

آبجکت‌ها

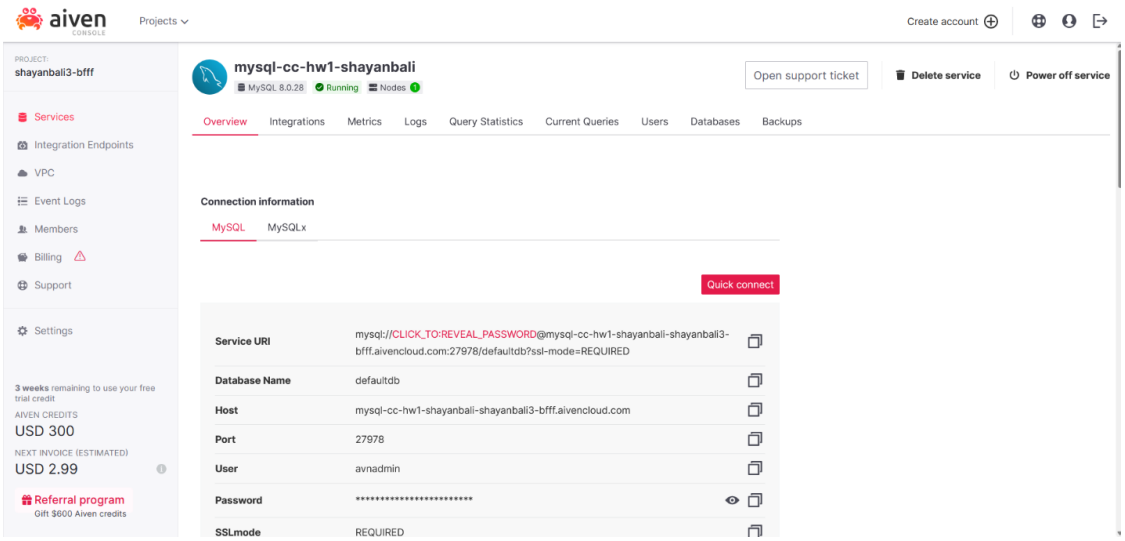
برای ساخت آبجکت‌های آگهی من از کلاس models خود جنگو استفاده کردم و فیلدهای آن را مطابق خواسته تمرین قرار دادم و فقط یک فیلد برای آدرس تصویر قرار دادم. همچنین وضعیت اولیه آگهی را نیز در حال بررسی (pending) قرار دادم.

شمای پیشنهادی سطرهای جدول آگهی‌ها در پایگاه داده:

id (int)	description (string)	email (string)	state (string)	category (string)

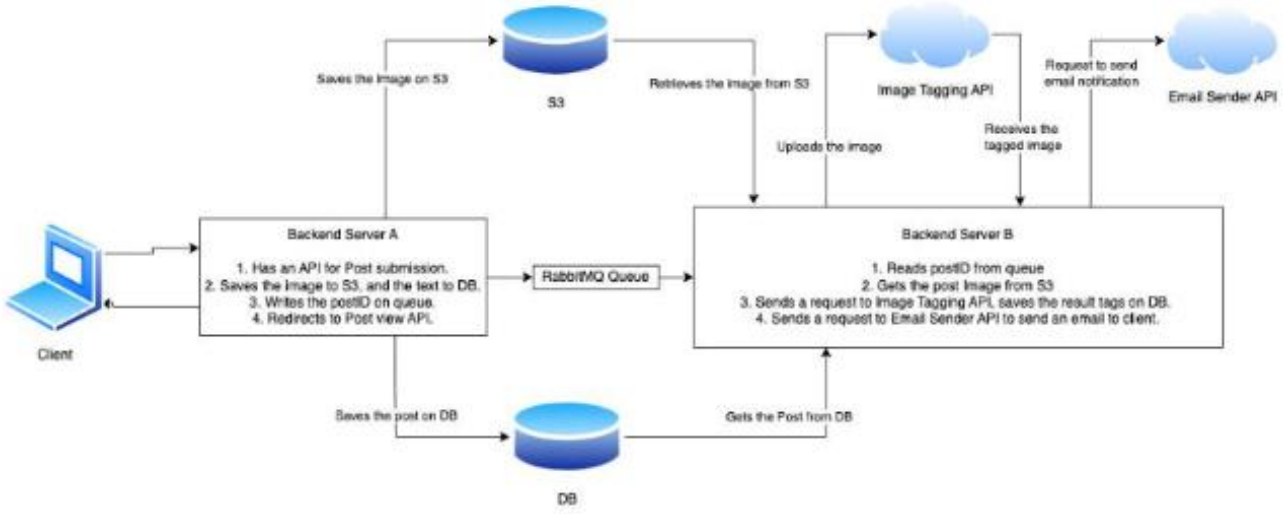
```python
class Ads(models.Model):
    description = models.CharField(max_length=400)
    email = models.CharField(max_length=100)
    state = models.CharField(max_length=100, blank=True, null=True, default='pending')
    category = models.CharField(max_length=100, blank=True, null=True)
    img = models.CharField(max_length=100, blank=True, null=True)
```

در گام بعد نیز سریالایلزر مرتبط با آبجکت را نیز انجام دادم و فیلد آیدی را در آن قرار دادم. برای پایگاه‌داده به عنوان سرویس نیز از سایت aiven کمک گرفتم و یک پایگاه داده در آن ایجاد کردم.



بعد از ایجاد آبجکت‌ها و پایگاه داده به سراغ پیاده‌سازی API ها می‌رویم. برای ایجاد API های درخواست شده در کلاس views جنگو دو تابع تعریف کردم که یکی برای ارسال آگهی و یکی برای دریافت آگهی است که در ادامه آن‌ها را توضیح می‌دهم. به طور کلی هم از معماری پیشنهاد شده در دستورکار استفاده کردم.

معماری پیشنهادی

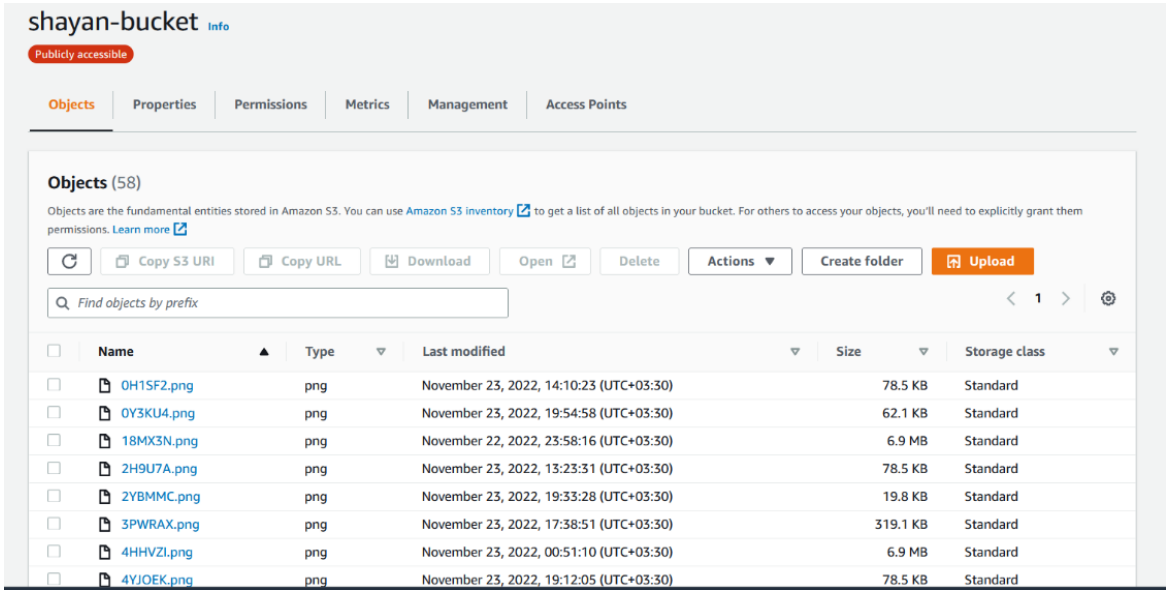


سرویس اول

تابع POST :

```python
def post(self, req):
    serializer = AdsSerializer(data=req.data)
    if not serializer.is_valid():
        return Response({"status": "BAD REQUEST", "data": serializer.errors}, status=status.HTTP_400_BAD_REQUEST)
    else:
        serializer.save()
        try:
            img = req.FILES.get('image', '')
            hash_fn = basehash.base36()
            hashed_id = hash_fn.hash(serializer.data['id'])
            img_id = str(hashed_id) + '.png'
            default_storage.save(img_id, img)
            img_url = bucket_url + img_id
            editUrl(img_url, serializer.data['id'])


        except:
            img = None

        sendID_rabbit(serializer.data['id'])
        return Response({"status": "successful", "data": serializer.data}, status=status.HTTP_200_OK)
```

این تابع در ویوهای ما کار API دریافت آگهی را هندل می‌کند. این تابع درخواست ثبت آگهی را به عنوان ریکوئست دریافت می‌کند و از آن به کمک سریالایلزر آبجکت می‌سازد و بعد چک می‌کند فرمت ریکوئست معتبر هست یا نه که اگر نباشد، پاسخ متناسب را می‌دهد. در مرحله بعد با استفاده از ترای، ۲ حالت عکس دار و بدون عکس بودن آگهی هندل می‌شود. در بدنه ترای دارای عکس بودن هندل می‌شود و سعی می‌کند از ریکوئست عکس را دریافت کند. بعد برای اسم عکس در ذخیره‌ساز شی، عدد آیدی را hash می‌کنم و به عنوان اسم قرار می‌دهم. برای ذخیره‌ساز شی هم از S3 استفاده کردم و یک باکت در آن ایجاد کردم. و آن را به عنوان ذخیره‌ساز پیش‌فرض در پروژه جنگو ام ست کردم.



در مرحله بعد لازم بود که لینک آبجکت در ذخیره‌ساز را در پایگاه داده قرار دهم که به کمک تابع editUrl صورت می‌گیرد.

```
def editUrl(img_url, imageId):

    mydb = mysql.connector.connect(
        host=HOST,
        port=PORT,
        user=USER,
        password=PASSWORD,
        database=DATABASE
    )
    mycursor = mydb.cursor(buffered=True)
    sql = 'UPDATE firstApi_ads SET img = (%s) WHERE id = (%s)'
    val = (img_url, imageId)
    mycursor.execute(sql, val)
    mydb.commit()
```

در این تابع که در فایل rabbitMQ است، ابتدا به پایگاه داده متصل می‌گردد و بعد فیلد آدرس عکس را با آدرس به دست آمده پر می‌کند. بخش except نیز برای مواقعی است که تصویر نیست ودر نتیجه فیلد آدرس عکس خالی می‌ماند. در آخر هم آیدی آگهی به صف rabbitMQ ارسال می‌گردد که برای این کار از تابع sendID_rabbit استفاده کردم که در ادامه آن را توضیح می‌دهم. همچنین برای صف rabbitMQ نیز از سایت cloudAMQP استفاده کردم.

```
# from tagging import tagImage
# from database import editState, sendMail

AMQP_URL = "amqps://yivdzdak:m8d9DdDZB9S5E_sNXeCMRVqMvgUg3vSI@toad.rmq.cloudamqp.com/yivdzdak"
bucket_url = 'https://shayan-bucket.s3.amazonaws.com/'

HOST = "mysql-cc-hw1-shayanbali-shayanbali3-bfff.aivencloud.com"
PORT = 27978
USER = "avnadmin"
PASSWORD = "AVNS_9XBFtm4jwXZQkSpUlig"
DATABASE = "defaultdb"


def sendID_rabbit(adID):
    connection = pika.BlockingConnection(pika.URLParameters(AMQP_URL))
    channel = connection.channel()
    channel.queue_declare(queue='ad_ids')
    channel.basic_publish(exchange='', routing_key='ad_ids', body=str(adID))
    connection.close()
```

همان‌طور که مشخص است تابع sendID_rabbit ، آیدی را دریافت می‌کند و بعد از اتصال به صف مورد نظر که در اینجا ad_ids می‌باشد، آیدی مدنظر را در صف برای ادامه کار قرار می‌دهد. این تابع نیز در فایل rabbitMQ قرار دارد.

در ادامه به بیان تابع GET جهت دریافت آگهی‌ها می‌پردازم.

تابع GET :

```python
class AdsViewSet(APIView):
    def get(self, req, *args, **kwargs):
        serializer = AdsSerializer(Ads.objects.get(id=req.data.get('id')))
        data = "Nothing"
        state = serializer.data.get('state')
        if state == 'confirmed':
            data = serializer.data
        res = {"AD_status": state, "AD_data": data}
        return Response(res, status=status.HTTP_200_OK)
```

در این تابع ریکوئست دریافت آگهی را می‌گیرد و به کمک فیلد آیدی در ریکوئست، آبجکت آن را با سریالایزر دریافت می‌کند و در صورتی که وضعیت آگهی تایید شده باشد، دیتای مرتبط با آن آگهی را برمی‌گرداند و اگر تایید نباشد چیزی برنمی‌گرداند.

سرویس دوم:

برای سرویس دوم من فایل دیگری به عنوان rabbitMQapi ساختم و همزمان با اجرای سرور، این فایل نیز اجرا می‌گردد که کار مربوط به سرویس دوم را اجرا کند. کار این سرویس خواندن آگهی از صف و بررسی آن است.

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

    channel.queue_declare(queue='ad_ids')

    def callback(ch, method, properties, body):
        print(" [x] Received %r" % body)
        print(body, "    body")
        # image url from s3
        imageID = str(body).replace('b', '').replace('\'', '')
        # hash_fn = basehash.base36()
        # hash_value = hash_fn.hash(13)
        # imageID_hashed = hash_fn.hash(imageID)
        # image_url = bucket_url + imageID_hashed + '.png'

        image_url = getURL_fromID(imageID)
        # send it to tagging system
        state, category = tagImage(image_url)
        editState(state, category, imageID)
        # send email
        req = sendMail(imageID, state)

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

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

این تابع به صف rabbitMQ که ساختیم متصل می‌شود و آیدی‌ها را از روی آن می‌خواند که در body قرار دارند. در مرحله بعد لازم است که آدرس عکس متناظر با آگهی به دست آید که به کمک تابع getURL_fromID صورت می‌گیرد که این تابع در فایل database قرار دارد.

```
def getURL_fromID(image_ID):
    mydb = mysql.connector.connect(
        host=HOST,
        port=PORT,
        user=USER,
        password=PASSWORD,
        database=DATABASE
    )
    mycursor = mydb.cursor(buffered=True)


    print(int(image_ID), "image id")
    sql = 'SELECT img from firstApi_ads where id = (%s)'
    val = (int(image_ID),)
    mycursor.execute(sql, val)
    image_url = mycursor.fetchall()
    print(image_url)
    final_URL = ""
    if len(image_url) > 0:
        final_URL = image_url[0][0]
    print(final_URL, "final url")
    return final_URL
```

این تابع به پایگاه داده ما متصل میگردد و بعد براساس آیدی، آدرس تصویر را برمی‌گرداند و اگر تصویر نداشته باشد خالی برمی‌گرداند.

بعد از دریافت آدرس تصویر نوبت تعیین وضعیت و دسته آگهی می‌گردد که برای تعیین آن از تابع tagImage در فایل database استفاده کردم. این تابع با دریافت URL و بررسی آن ، وضعیت و دسته‌ی آگهی را تعیین می‌کند.

```python
def tagImage(IMAGE_URL):
    state = 'rejected'
    category = ''
    print(IMAGE_URL, "Image URL")
    if IMAGE_URL is not None:

        response = requests.get(
            'https://api.imagga.com/v2/tags?image_url=%s' % IMAGE_URL, auth=(API_KEY, API_SECRET))

        tags = response.json()
        tags = tags['result']['tags']
        for tag in tags:
            if tag['tag']['en'] == 'vehicle' and tag['confidence'] > 50:
                state = 'confirmed'

        category = tags[0]['tag']['en']

    return state, category
```

این تابع چک می‌کند که در ابتدا اگر آدرس خالی باشد، یعنی آگهی عکسی نداشته باشد، آن را رد می‌کند؛ اما اگر عکس داشت چک می‌کند که آیا عکس مربوط به وسیله نقلیه هست یا نه و اگر نبود آن را رد می‌کند (براساس شرط ذکر شده در صورت تمرین). بعد از تایید کردن آگهی براساس بیشترین میزان تگ، دسته آگهی را تعیین می‌کند و در نهایت وضعیت آگهی و دسته آن به عنوان خروجی داده می‌شود. کل این پروسه به کمک سایت برچسب‌زنی imagga صورت می‌گیرد. بعد از آن نوبت آن می‌رسد که وضعیت و دسته به دست آمده را در پایگاه داده به روز کنیم که این کار به کمک تابع editState در فایل database صورت می‌گیرد .

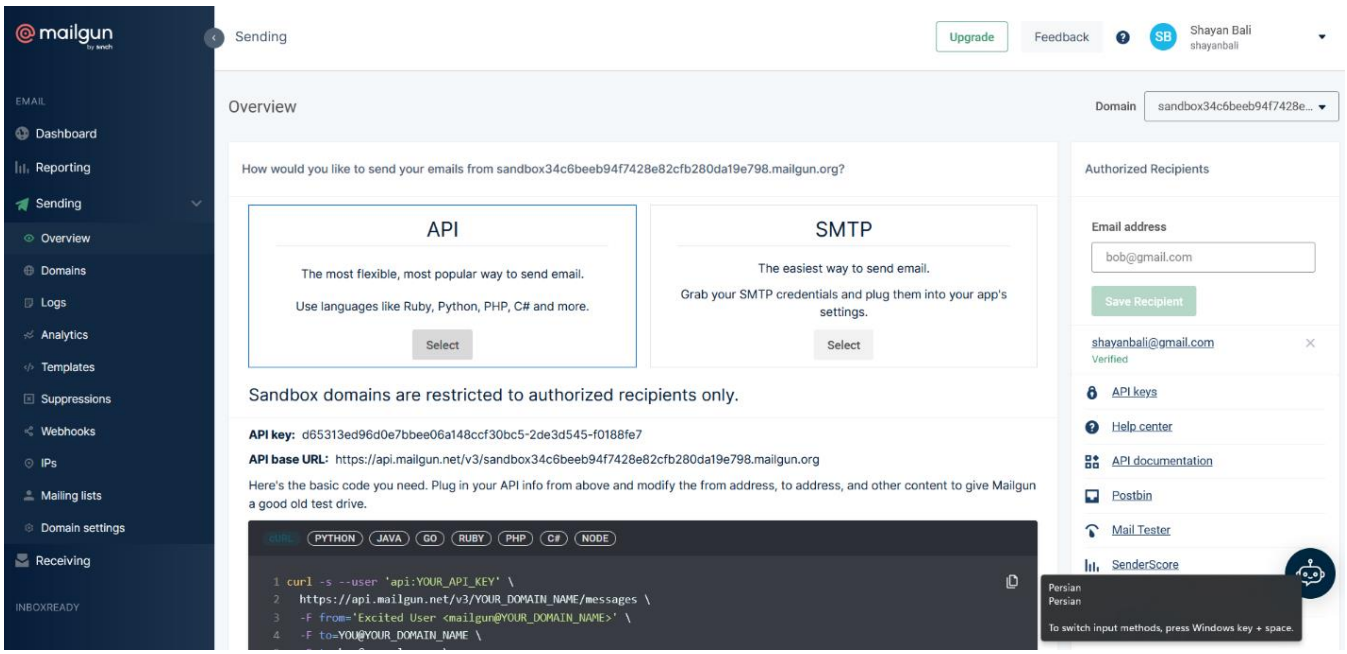
```python
def editState(state, category, imageId):
    mydb = mysql.connector.connect(
        host=HOST,
        port=PORT,
        user=USER,
        password=PASSWORD,
        database=DATABASE
    )
    mycursor = mydb.cursor(buffered=True)
    sql = 'UPDATE firstApi_ads SET state = (%s), category = (%s) WHERE id = (%s)'
    val = (state, category, imageId)
    mycursor.execute(sql, val)

    mydb.commit()
```

این تابع به پایگاه داده متصل می‌شود و بعد براساس آیدی به آبجکت مورد نظر می‌رسد و فیلدهای وضعیت و دسته‌بندی آن را تغییر می‌دهد.

در نهایت هم براساس آخرین تغییرات و وضعیت آگهی ، ایمیل به کاربر ارسال می‌گردد که این کار با تابع sendMail صورت می‌گیرد. این تابع وضعیت و آیدی را می‌گیرد و براساس آن ایمیل را ارسال می‌کند که این تابع نیز در فایل database قرار دارد.

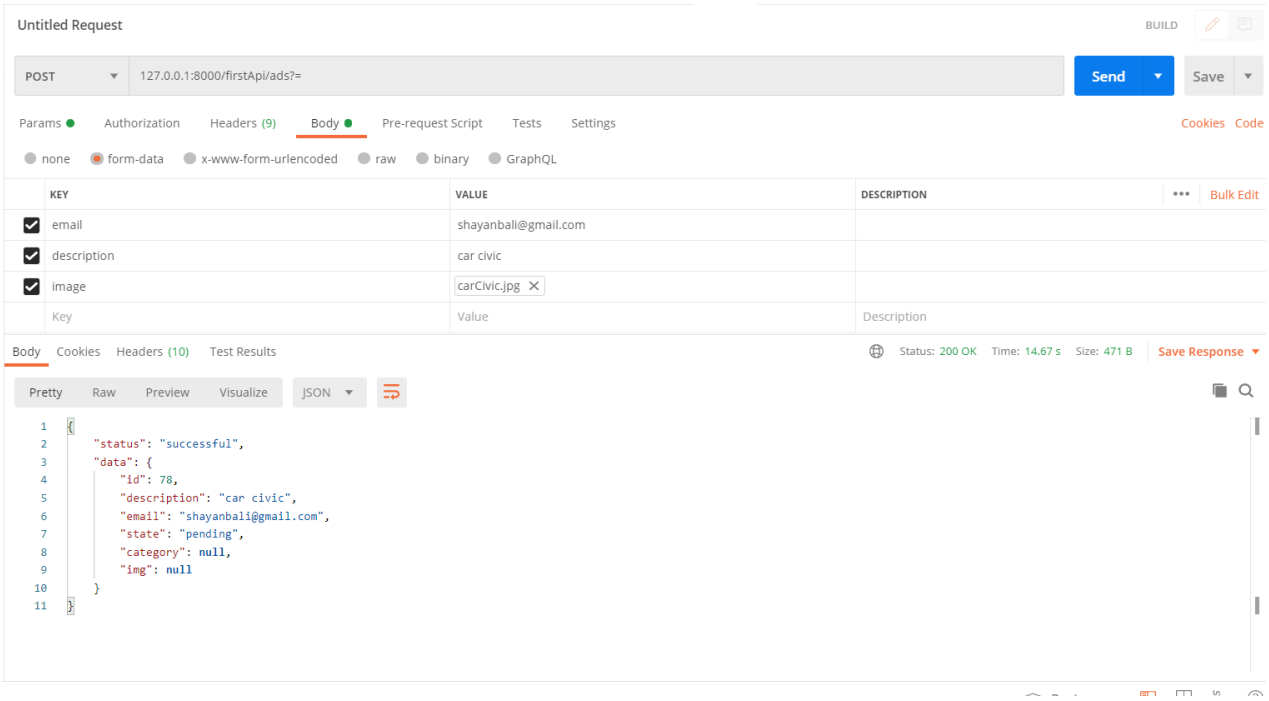
```python
def sendMail(id, state):
    mydb = mysql.connector.connect(
        host=HOST,
        port=PORT,
        user=USER,
        password=PASSWORD,
        database=DATABASE
    )
    mycursor = mydb.cursor(buffered=True)
    sql = 'SELECT email from firstApi_ads where id = (%s)'
    val = (id,)
    mycursor.execute(sql, val)
    email = mycursor.fetchall()
    email = email[0][0]
    extra_info = "\n"
    if state == 'confirmed':
        sql = 'SELECT category from firstApi_ads where id = (%s)'
        val = (id,)
        mycursor.execute(sql, val)
        category = mycursor.fetchall()
        category = category[0][0]
        extra_info += ("category: " + category + "\n")

        sql = 'SELECT description from firstApi_ads where id = (%s)'
        val = (id,)
        mycursor.execute(sql, val)
        description = mycursor.fetchall()
        description = description[0][0]
        extra_info += ("description: " + description + "\n")

    sql = 'SELECT img from firstApi_ads where id = (%s)'
    val = (id,)
    mycursor.execute(sql, val)
    img = mycursor.fetchall()
    img = img[0][0]
    extra_info += ("imageURL: " + img + "\n")

    return requests.post(
        "https://api.mailgun.net/v3/sandbox34c6beeb94f7428e82cfb280da19e798.mailgun.org/messages",
        auth=("api", "d65313ed96d0e7bbee06a148ccf30bc5-2de3d545-f0188fe7"),
        data={"from": "Mailgun Sandbox <postmaster@sandbox34c6beeb94f7428e82cfb280da19e798.mailgun.org>",
              "to": [email],
              "subject": 'Advertisement state',
              "text": 'advertisement id : ' + id + '\n the AD has been ' + state+extra_info})
```

این تابع ابتدا به پایگاه داده متصل می‌گردد و بعد ایمیل کاربر را از پایگاه داده می‌گیرد. بعد از آن اگر آگهی تایید شده بود مشخصات آن را از پایگاه داده می‌گیرد و در نهایت هم ارسال ایمیل صورت می‌گیرد که ارسال ایمیل نیز به کمک mailGun صورت می‌گیرد.

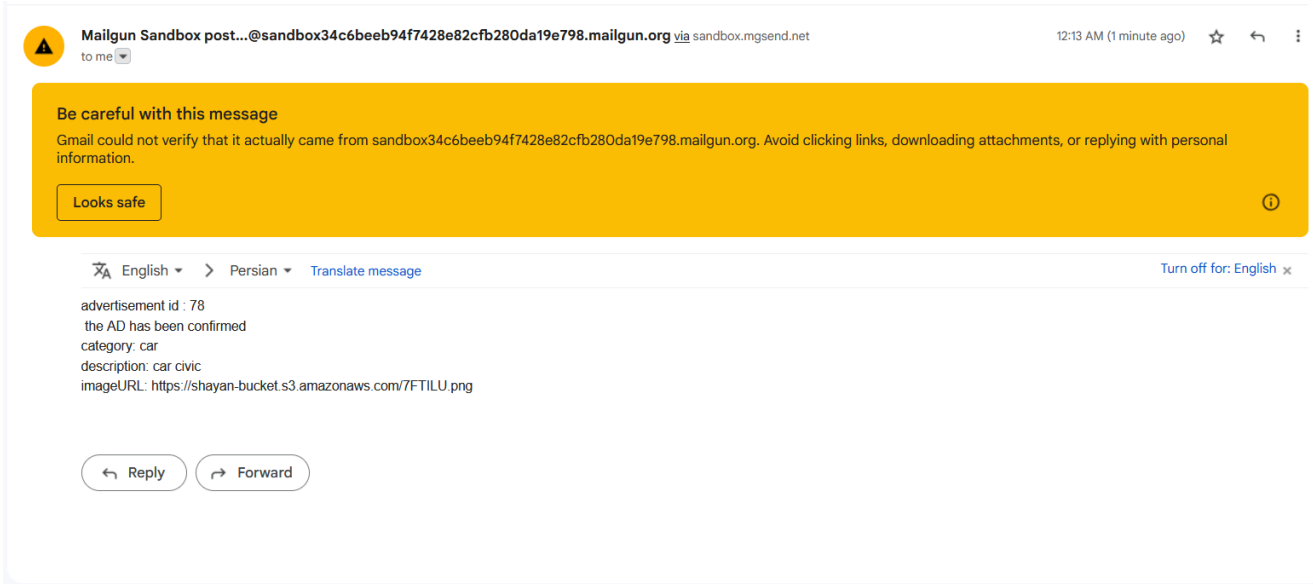


در ادامه مثالی از دستورات را قرار می‌دهم که کار کردن تمامی این توابع در کنار هم را نشان می‌دهند.

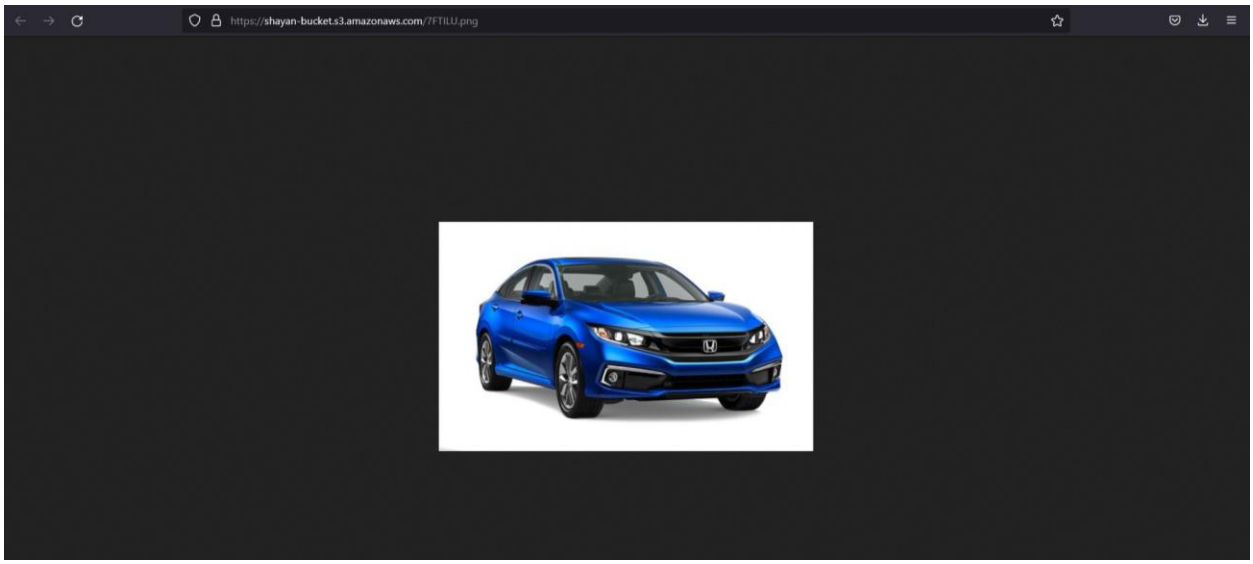
تست POST :



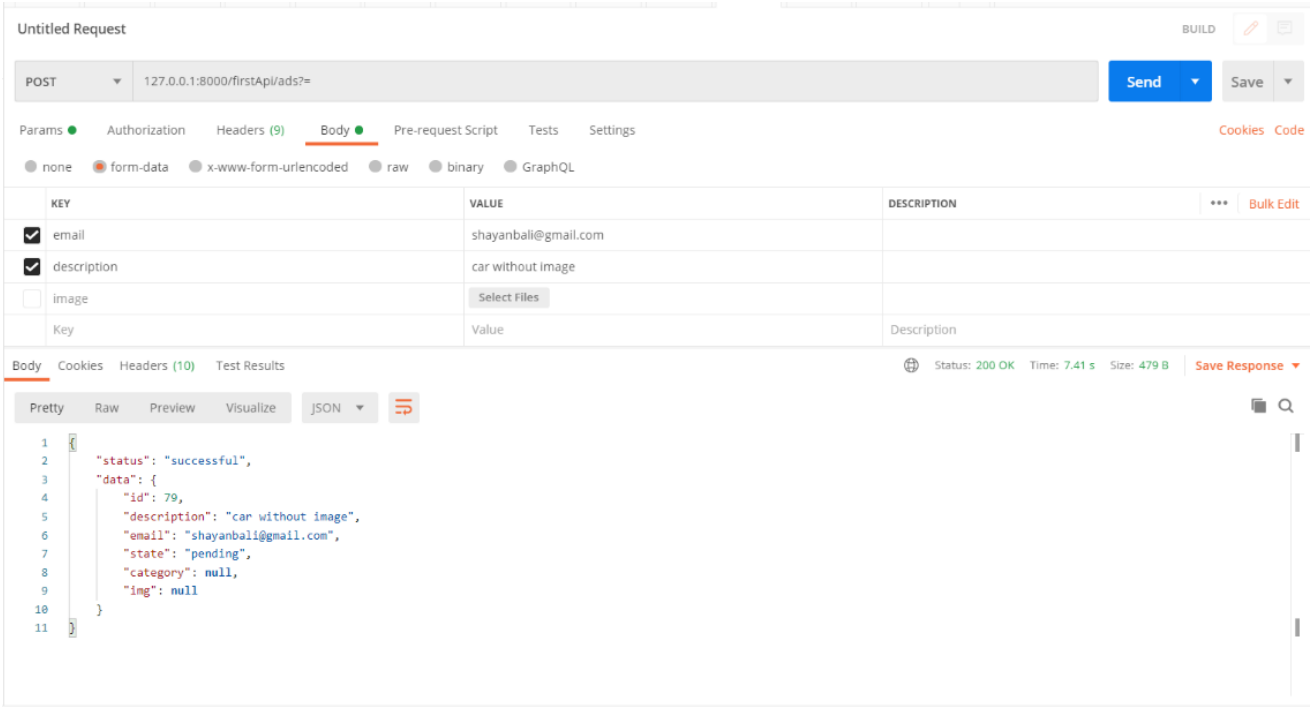
نتیجه :



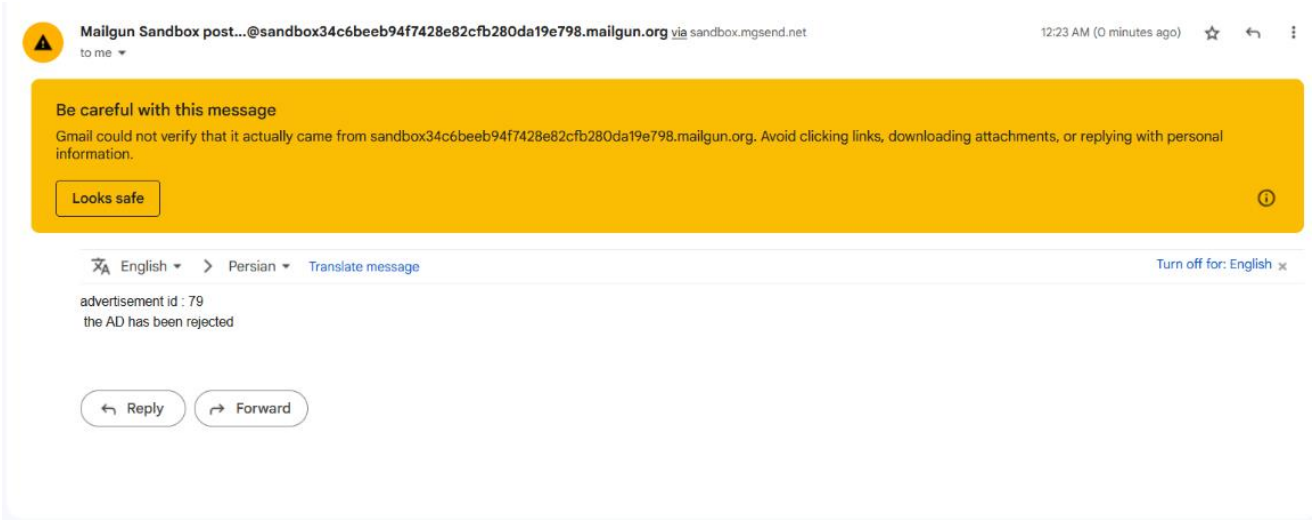
تصویر آگهی:



تست POST با آگهی بدون تصویر:



نتیجه:



تست GET: