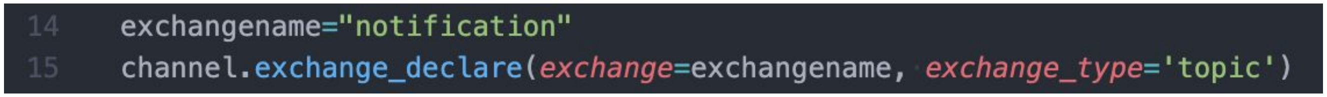
ESD G2-T4 FINAL PRESENTATION Q&A

Dear Prof and Eng Kit,
We will like to start off by saying we are truly apologetic for the video produced. We are very regretful that we are unable to showcase the entirety of our hard work as we ran into too many issues after deployment. No doubt the deadline was extended, but we had multiple deadlines and final exams falling on the same week the submission was due. This is no excuse for the presentation video but we hope this Q&A and our report would provide both of you with a more indepth insight as to how our application would work.

Questions

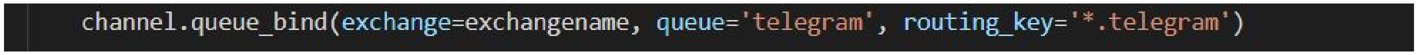
1. **What kind of exchange(s) did you use on your RabbitMQ broker, and how were messages routed to your Email/Telegram notification services?**

   In Notification.py, we utilised Topic exchange for the RabbitMQ broker,

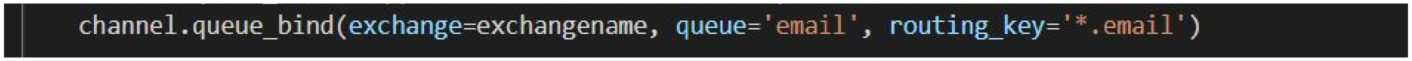


```
exchangename="notification"
channel.exchange_declare(exchange=exchangename, exchange_type='topic')
```

   and the messages were routed using a wildcard, '*', such as the following:



```
channel.queue_bind(exchange=exchangename, queue='telegram', routing_key='*.telegram')
```

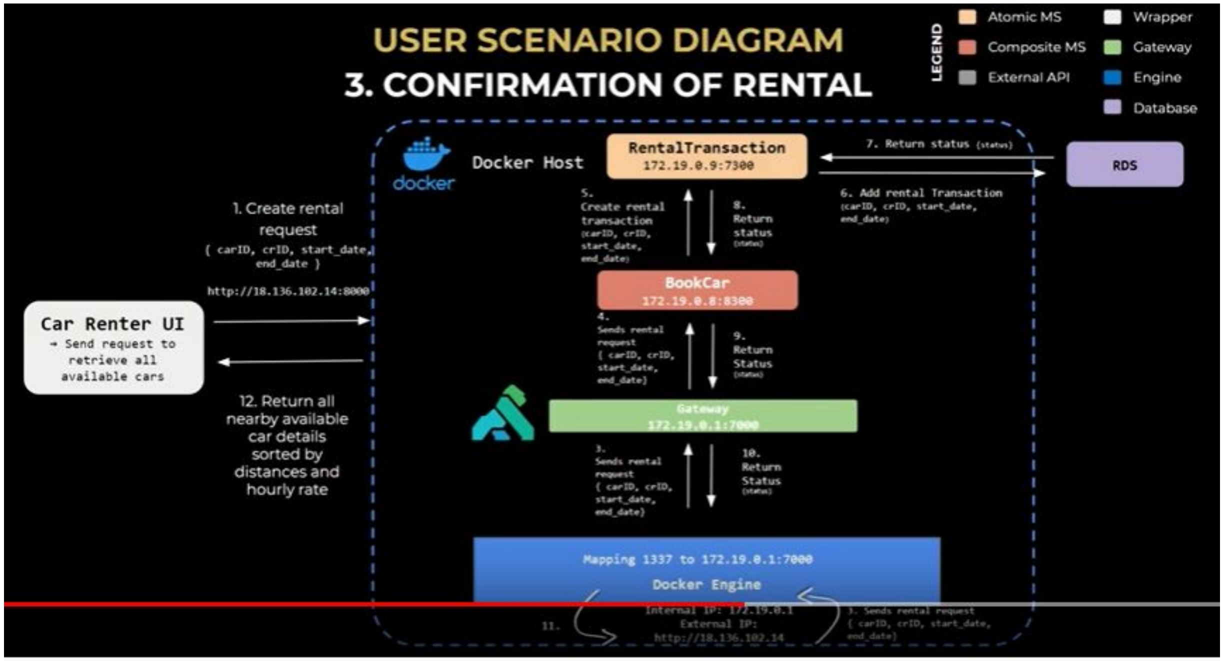


```
channel.queue_bind(exchange=exchangename, queue='email', routing_key='*.email')
```

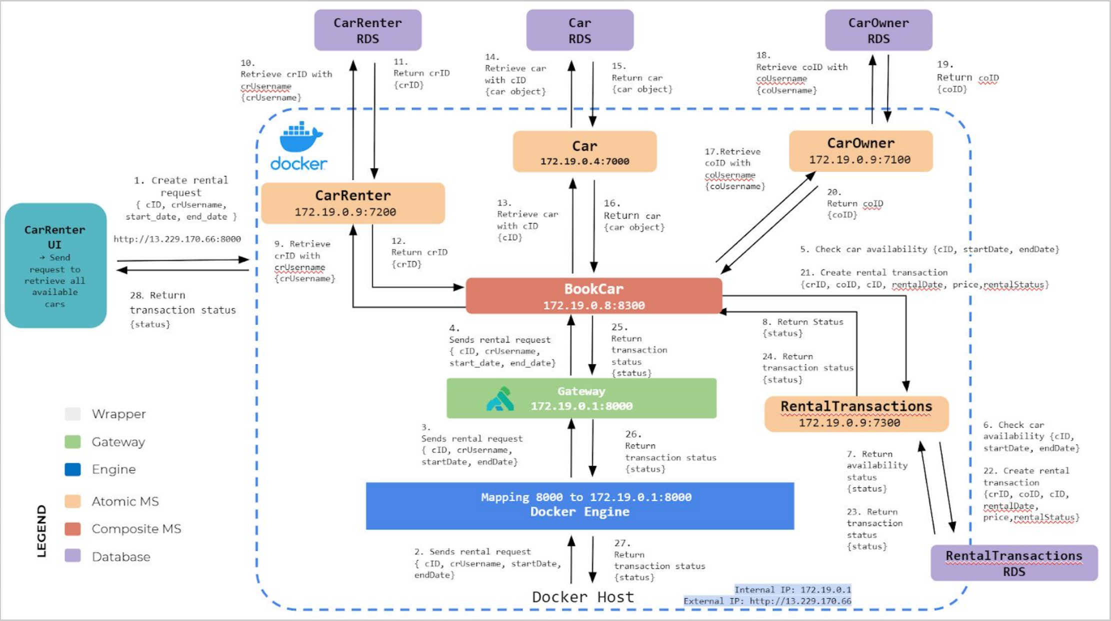
   The communication pattern is one-to-many, fire and forget.

   The reason behind the utilisation of topic exchange is to facilitate any new changes that could be added to this microservice in the future, as direct and fanout exchanges can be simulated easily with topic exchange.

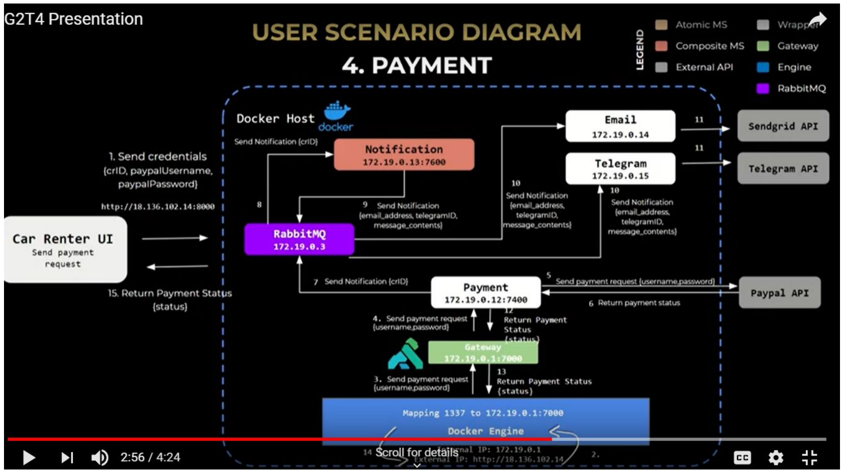
2. **For the report, ensure the protocols are indicated in all the user scenario diagrams**
   On this part, we have made necessary changes that are reflected in the report.


USER SCENARIO DIAGRAM
3. CONFIRMATION OF RENTAL
LEGEND
Atomic MS
Composite MS
External API
Wrapper
Gateway
Engine
Database
docker
Docker Host
RentalTransaction
172.19.0.9:7300
7. Return status (status)
RDS
6. Add rental Transaction (carID, crID, start_date, end_date)
5. Create rental transaction (carID, crID, start_date, end_date)
8. Return status (status)
BookCar
172.19.0.8:8300
4. Sends rental request { carID, crID, start_date, end_date}
9. Return Status (status)
Gateway
172.19.0.1:7000
3. Sends rental request { carID, crID, start_date, end_date}
10. Return Status (status)
Mapping 1337 to 172.19.0.1:7000
Docker Engine
Internal IP: 172.19.0.1
External IP: http://18.136.102.14
11.
3. Sends rental request { carID, crID, start_date, end_date}
1. Create rental request
{ carID, crID, start_date, end_date }
http://18.136.102.14:8000
Car Renter UI
→ Send request to retrieve all available cars
12. Return all nearby available car details sorted by distances and hourly rate

3. **Why did you use a composite (BookCar) service when only 1 other service (RentalTransaction) is called?**
   BookCar composite service interacts with rentalTransactions, Car, CarRenter and CarOwner atomic microservices. It interacts with the Car microservice to retrieve carID, CarRenter microservice to retrieve the CarRenter's IDand CarOwner microservice to retrieve the respective car's CarOwner's ID. All of the above information will subsequently be used to create the rental transaction. We left this out in the slides and did not indicate that it interacts with these three other services. We will be indicating it clearly in the report for the final submission. Below is a screenshot of the updated scenario:



Screenshot of user scenario 3: Car Renter selects Car for Rental



G2T4 Presentation

USER SCENARIO DIAGRAM

4. PAYMENT

LEGEND

🟫 Atomic MS
🟥 Composite MS
⬜ External API
⬜ Wrapper
🟩 Gateway
🟦 Engine
🟪 RabbitMQ

Docker Host

Send Notification (crID)

Notification
172.19.0.13:7600

Email
172.19.0.14

Telegram
172.19.0.15

11

11

Sendgrid API

Telegram API

1. Send credentials (crID, paypalUsername, paypalPassword)

http://18.136.102.14:8000

Car Renter UI
Send payment request

15. Return Payment Status (status)

8

RabbitMQ
172.19.0.3

9 Send Notification (email_address, telegramID, message_contents)

10 Send Notification (email_address, telegramID, message_contents)

10 Send Notification (email_address, telegramID, message_contents)

7 Send Notification (crID)

Payment
172.19.0.12:7400

5 Send payment request (username,password)

6 Return payment status

Paypal API

4. Send payment request (username,password)

12 Return Payment Status (status)

Gateway
172.19.0.1:7000

3. Send payment request (username,password)

13 Return Payment Status (status)

Mapping 1337 to 172.19.0.1:7000

Docker Engine

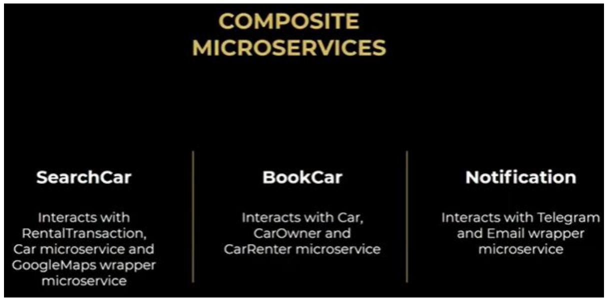
14

Internal IP: 172.19.0.1

External IP: http://18.136.102.14

2.

2:56 / 4:24
Scroll for details



COMPOSITE MICROSERVICES

SearchCar

Interacts with RentalTransaction, Car microservice and GoogleMaps wrapper microservice

BookCar

Interacts with Car, CarOwner and CarRenter microservice

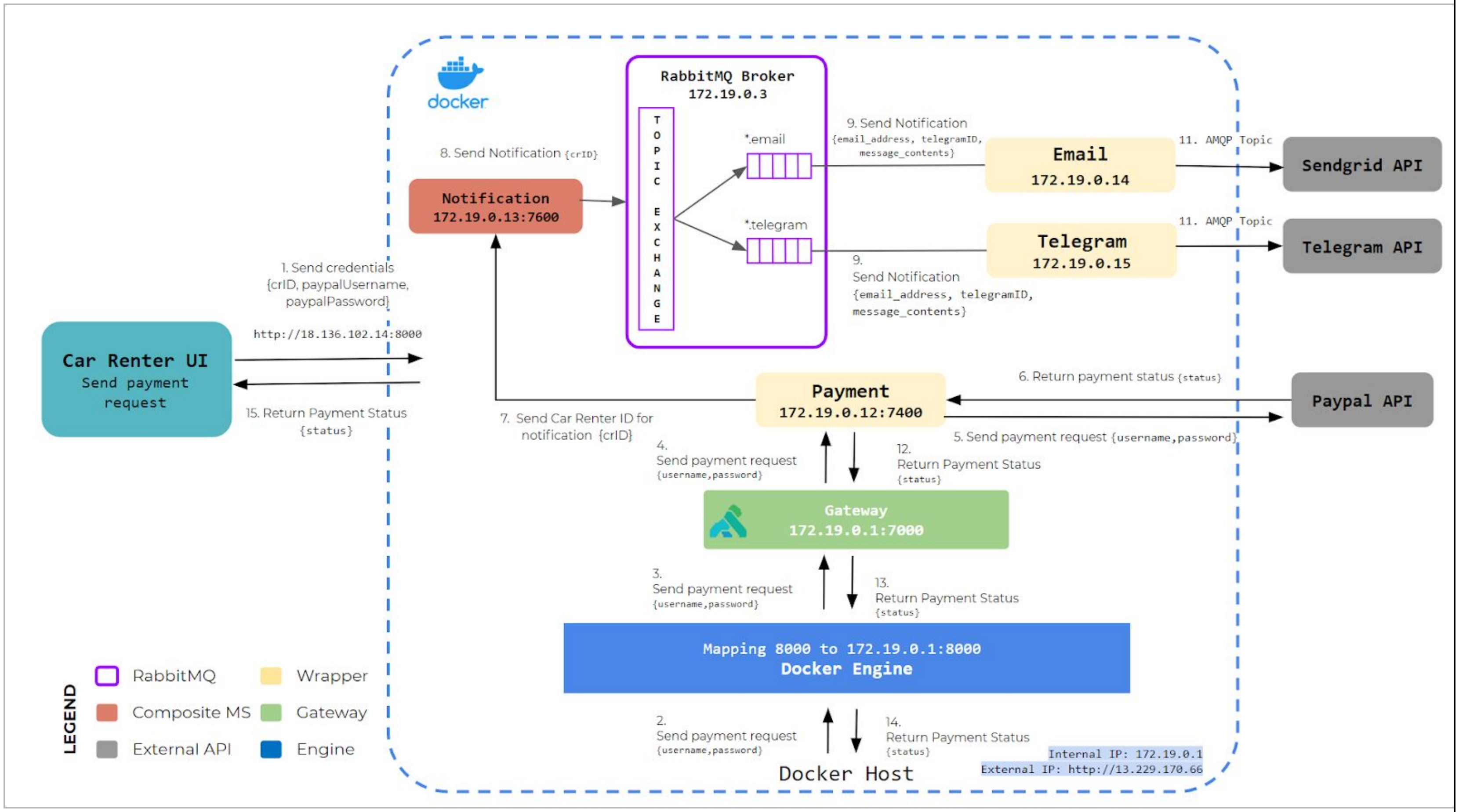
Notification

Interacts with Telegram and Email wrapper microservice

4. **Does the Notification service interact with the Email and Telegram wrapper as mentioned? Else, the name used does not really reflect the function.**

   The notification service interacts with Email and Telegram wrappers via the RabbitMQ broker's topic exchange. We chose to use this microservice instead of directly interacting with the email and telegram wrappers as the team felt that this enhances the reusability of the service for when the team wants to add an additional way of sending notification customers in the future.

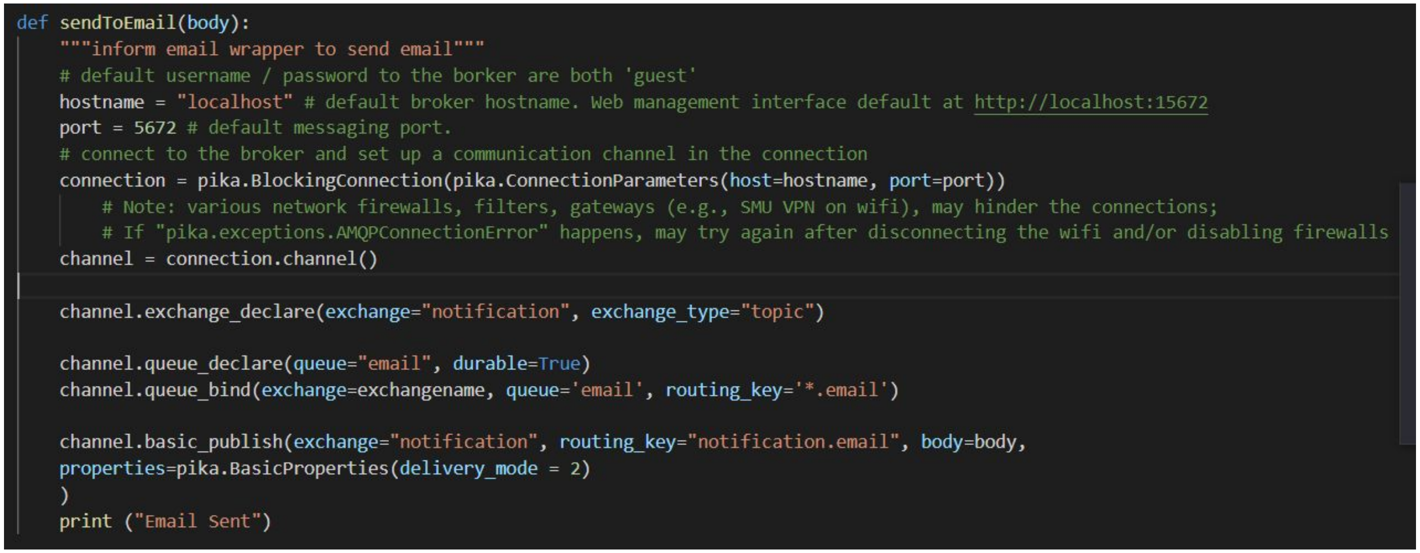
We have updated the representation of this in the final report with the following screenshot:



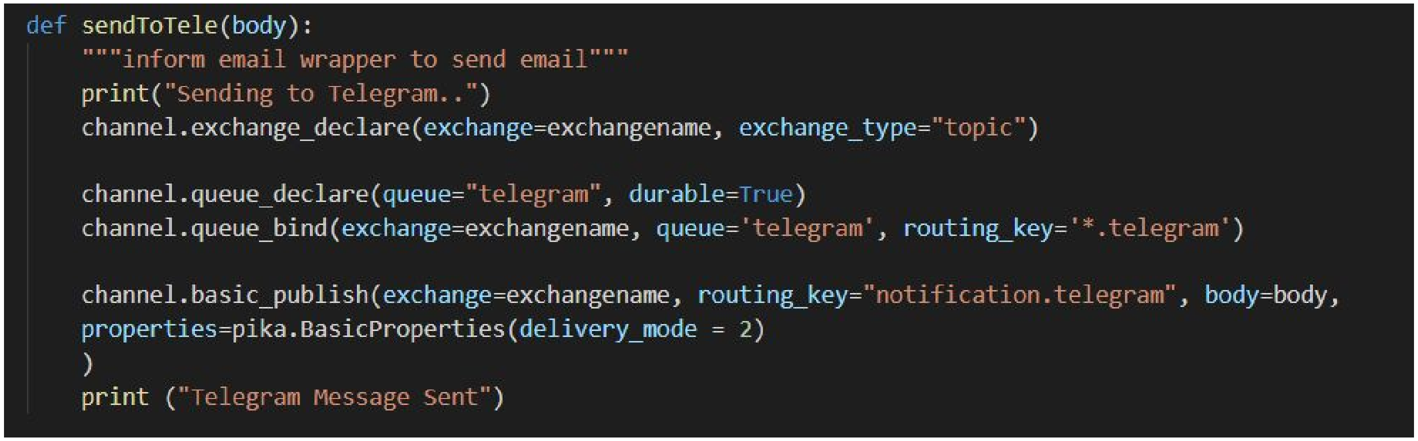
User scenario 4: Car Renter Makes Payment

This shows how the Notification microservice publishes messages to the RabbitMQ broker's topic exchange which will then route the messages to the queues with matching routing patterns accordingly. The Email and Telegram wrapper microservices will receive the messages accordingly and trigger the external API.

The interaction is as shown in the codes screenshot below, from Notification.py:

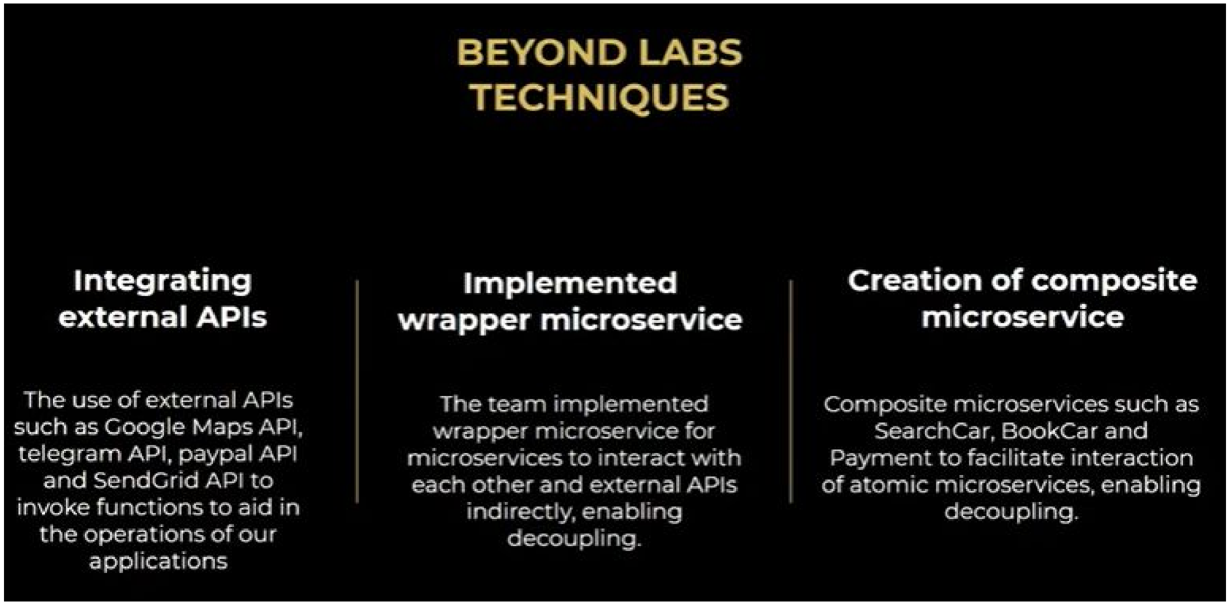


```
def sendToEmail(body):
    """inform email wrapper to send email"""
    # default username / password to the borker are both 'guest'
    hostname = "localhost" # default broker hostname. Web management interface default at http://localhost:15672
    port = 5672 # default messaging port.
    # connect to the broker and set up a communication channel in the connection
    connection = pika.BlockingConnection(pika.ConnectionParameters(host=hostname, port=port))
        # Note: various network firewalls, filters, gateways (e.g., SMU VPN on wifi), may hinder the connections;
        # If "pika.exceptions.AMQPConnectionError" happens, may try again after disconnecting the wifi and/or disabling firewalls
    channel = connection.channel()

    channel.exchange_declare(exchange="notification", exchange_type="topic")

    channel.queue_declare(queue="email", durable=True)
    channel.queue_bind(exchange=exchangename, queue='email', routing_key='*.email')

    channel.basic_publish(exchange="notification", routing_key="notification.email", body=body,
    properties=pika.BasicProperties(delivery_mode = 2)
    )
    print ("Email Sent")
```



```
def sendToTele(body):
    """inform email wrapper to send email"""
    print("Sending to Telegram..")
    channel.exchange_declare(exchange=exchangename, exchange_type="topic")

    channel.queue_declare(queue="telegram", durable=True)
    channel.queue_bind(exchange=exchangename, queue='telegram', routing_key='*.telegram')

    channel.basic_publish(exchange=exchangename, routing_key="notification.telegram", body=body,
    properties=pika.BasicProperties(delivery_mode = 2)
    )
    print ("Telegram Message Sent")
```

5. **What does Notification do? Are there atomic services it calls?**
   Notification is responsible for sending notifications out to Car Renters and Car Owners after a successful rental transaction has been created (i.e., after successful payment has been made). The notification composite microservice will publish messages to the RabbitMQ broker with the corresponding routing key to match the binding patterns for the respective queues that the wrapper microservices Email and Telegram will consume from.



BEYOND LABS TECHNIQUES

Integrating external APIs

The use of external APIs such as Google Maps API, telegram API, paypal API and SendGrid API to invoke functions to aid in the operations of our applications

Implemented wrapper microservice

The team implemented wrapper microservice for microservices to interact with each other and external APIs indirectly, enabling decoupling.

Creation of composite microservice

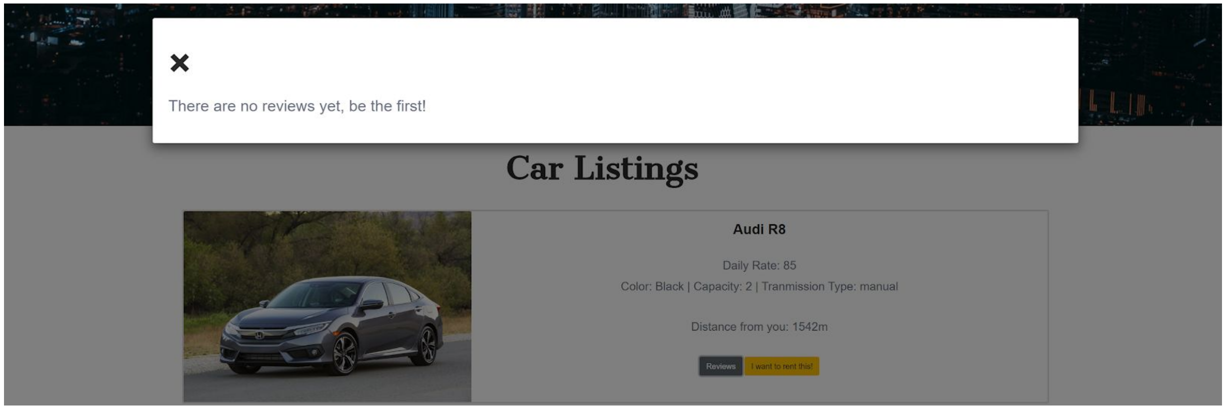
Composite microservices such as SearchCar, BookCar and Payment to facilitate interaction of atomic microservices, enabling decoupling.

6. **What happened to Notification service under Composite?**
   In this slide, we mistakenly insert Payment instead of Notification in this slide under “Creation of composite microservice”. This was an oversight on our part and we are sorry about that.

7. In the demo, you mentioned reviews… but it does not appear in the user scenario diagrams at all.

Reviews was created and works but we did not put it into the user scenario diagram as the team felt that it is not significant enough to be its own scenario. It also does not fit into our existing user scenarios.

Upon keying in the search criteria, the users will be able to see a list of available cars and check out the car's reviews before booking the car.
When there are no reviews for the car, the following will be shown:



There are no reviews yet, be the first!

Car Listings

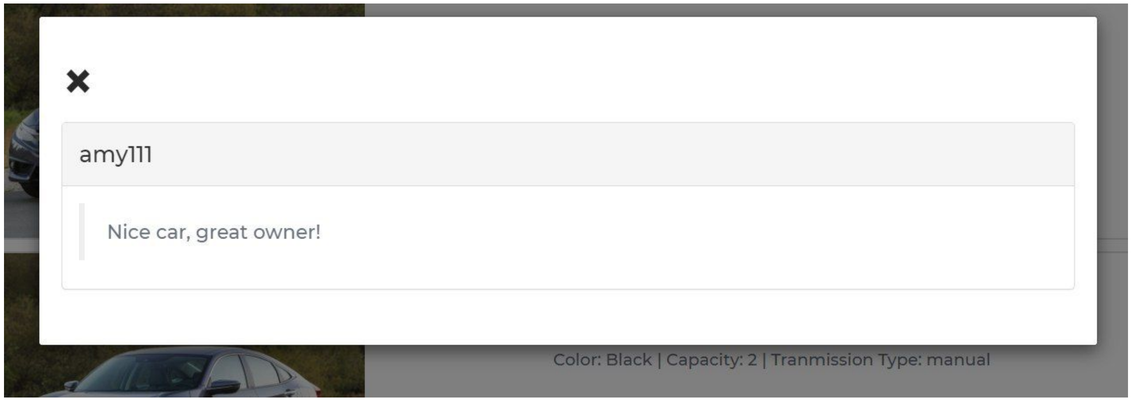
Audi R8

Daily Rate: 85

Color: Black | Capacity: 2 | Tranmission Type: manual

Distance from you: 1542m

For cars with reviews, upon clicking on the "reviews" button, it will display a pop up screen with the reviews written by users regarding the car.



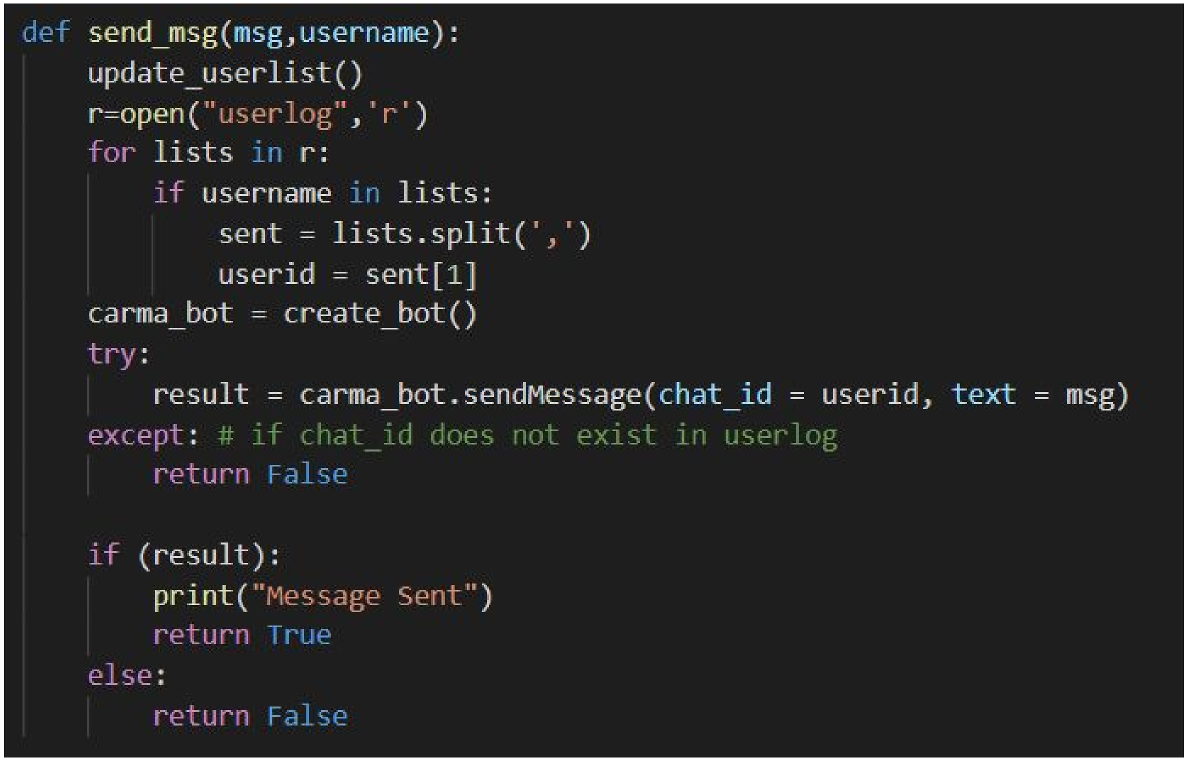
amy111

Nice car, great owner!

Color: Black | Capacity: 2 | Tranmission Type: manual

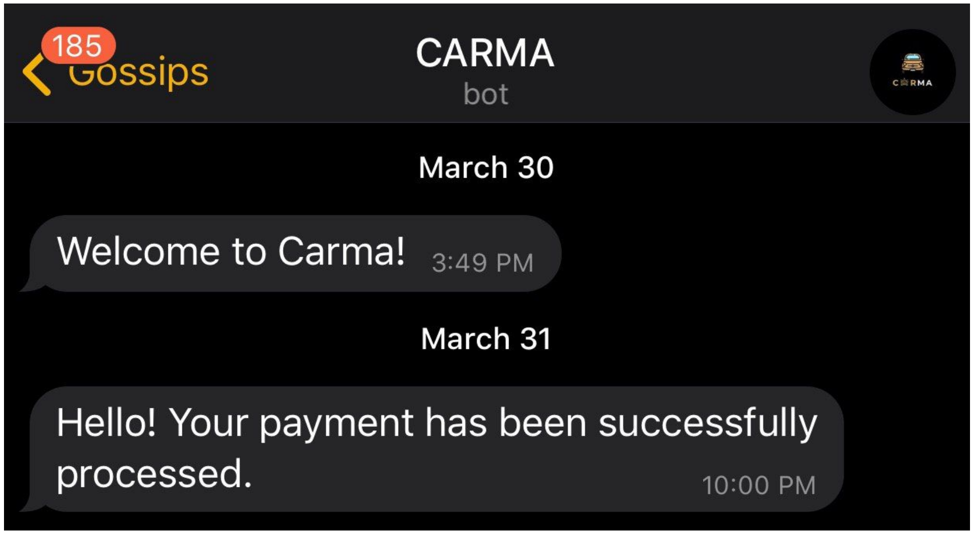
8. **Can you show evidence of those beyond the labs techniques that you have mentioned but not shown in the video? E.g. but not exhaustive: Docker compose, AWS, SendGrid, Google Maps, Kong (show key lines of code, screenshots, etc).**

Telegram API: refer to the submitted code for the codes we used to implement Telegram API in the notifications folder, in the bot.py file.



```python
def send_msg(msg,username):
    update_userlist()
    r=open("userlog",'r')
    for lists in r:
        if username in lists:
            sent = lists.split(',')
            userid = sent[1]
    carma_bot = create_bot()
    try:
        result = carma_bot.sendMessage(chat_id = userid, text = msg)
    except: # if chat_id does not exist in userlog
        return False

    if (result):
        print("Message Sent")
        return True
    else:
        return False
```

Evidence of Telegram API working as shown:



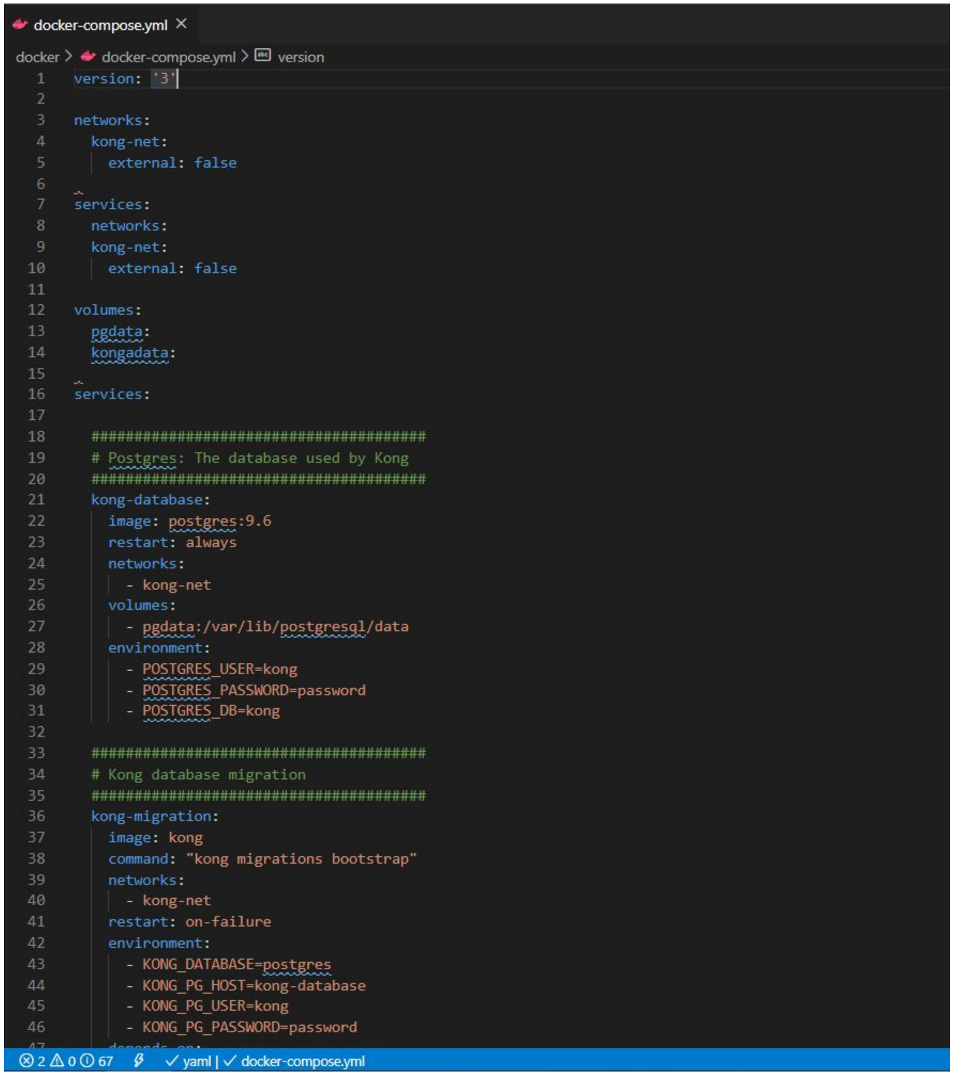
March 30

Welcome to Carma! 3:49 PM

March 31

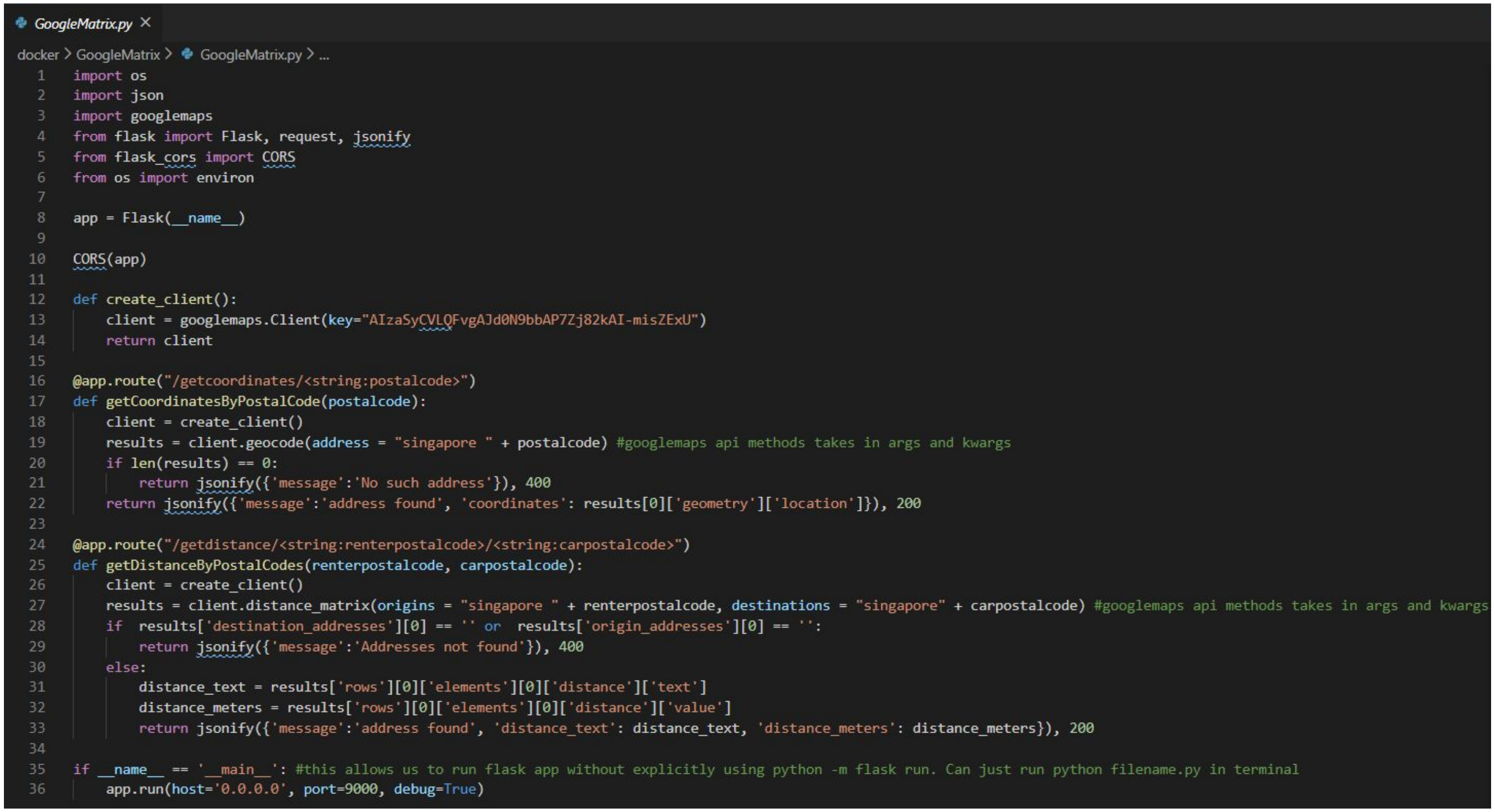
Hello! Your payment has been successfully processed. 10:00 PM

Docker-Compose: refer to the submitted code for the full version located in docker/docker-compose.yml


docker-compose.yml ×
docker > docker-compose.yml > version
⊗ 2 ⚠ 0 ⓘ 67
✓ yaml | ✓ docker-compose.yml

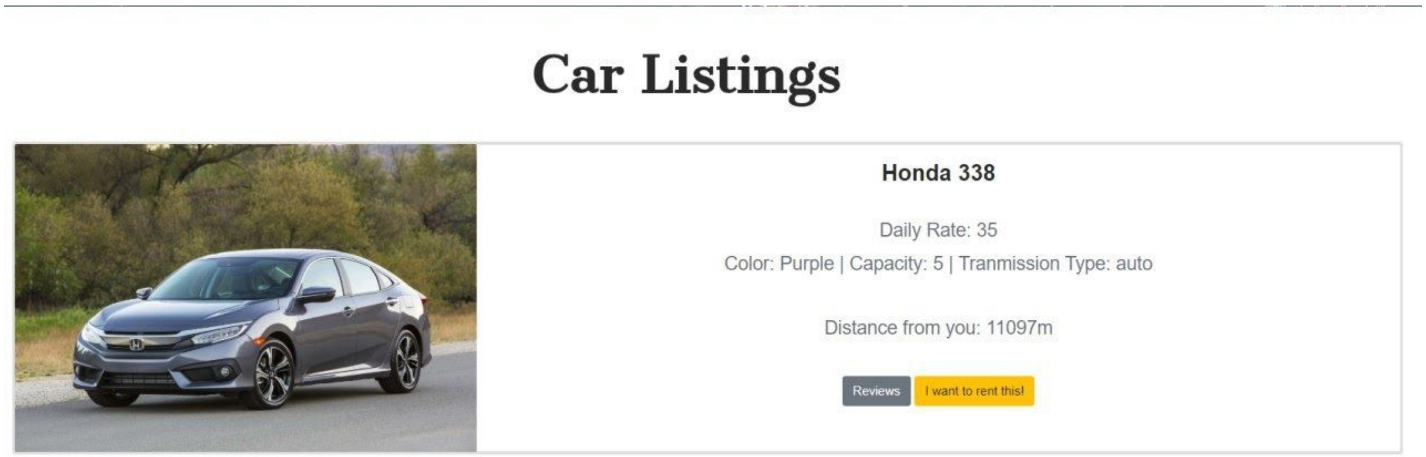
```yaml
version: '3'

networks:
  kong-net:
    external: false

services:
  networks:
  kong-net:
    external: false

volumes:
  pgdata:
  kongadata:

services:

  #######################################
  # Postgres: The database used by Kong
  #######################################
  kong-database:
    image: postgres:9.6
    restart: always
    networks:
      - kong-net
    volumes:
      - pgdata:/var/lib/postgresql/data
    environment:
      - POSTGRES_USER=kong
      - POSTGRES_PASSWORD=password
      - POSTGRES_DB=kong

  #######################################
  # Kong database migration
  #######################################
  kong-migration:
    image: kong
    command: "kong migrations bootstrap"
    networks:
      - kong-net
    restart: on-failure
    environment:
      - KONG_DATABASE=postgres
      - KONG_PG_HOST=kong-database
      - KONG_PG_USER=kong
      - KONG_PG_PASSWORD=password
    depends_on:
```

Google Maps API: refer to the submitted code for full version located in docker/GoogleMatrix/GoogleMatrix.py



```python
import os
import json
import googlemaps
from flask import Flask, request, jsonify
from flask_cors import CORS
from os import environ

app = Flask(__name__)

CORS(app)

def create_client():
    client = googlemaps.Client(key="AIzaSyCVLQFvgAJd0N9bbAP7Zj82kAI-misZExU")
    return client

@app.route("/getcoordinates/<string:postalcode>")
def getCoordinatesByPostalCode(postalcode):
    client = create_client()
    results = client.geocode(address = "singapore " + postalcode) #googlemaps api methods takes in args and kwargs
    if len(results) == 0:
        return jsonify({'message':'No such address'}), 400
    return jsonify({'message':'address found', 'coordinates': results[0]['geometry']['location']}), 200

@app.route("/getdistance/<string:renterpostalcode>/<string:carpostalcode>")
def getDistanceByPostalCodes(renterpostalcode, carpostalcode):
    client = create_client()
    results = client.distance_matrix(origins = "singapore " + renterpostalcode, destinations = "singapore" + carpostalcode) #googlemaps api methods takes in args and kwargs
    if  results['destination_addresses'][0] == '' or  results['origin_addresses'][0] == '':
        return jsonify({'message':'Addresses not found'}), 400
    else:
        distance_text = results['rows'][0]['elements'][0]['distance']['text']
        distance_meters = results['rows'][0]['elements'][0]['distance']['value']
        return jsonify({'message':'address found', 'distance_text': distance_text, 'distance_meters': distance_meters}), 200

if __name__ == '__main__': #this allows us to run flask app without explicitly using python -m flask run. Can just run python filename.py in terminal
    app.run(host='0.0.0.0', port=9000, debug=True)
```

Upon keying in search criteria for cars available, the UI shows the distance between the user's input location and the car's location, as shown:



Car Listings

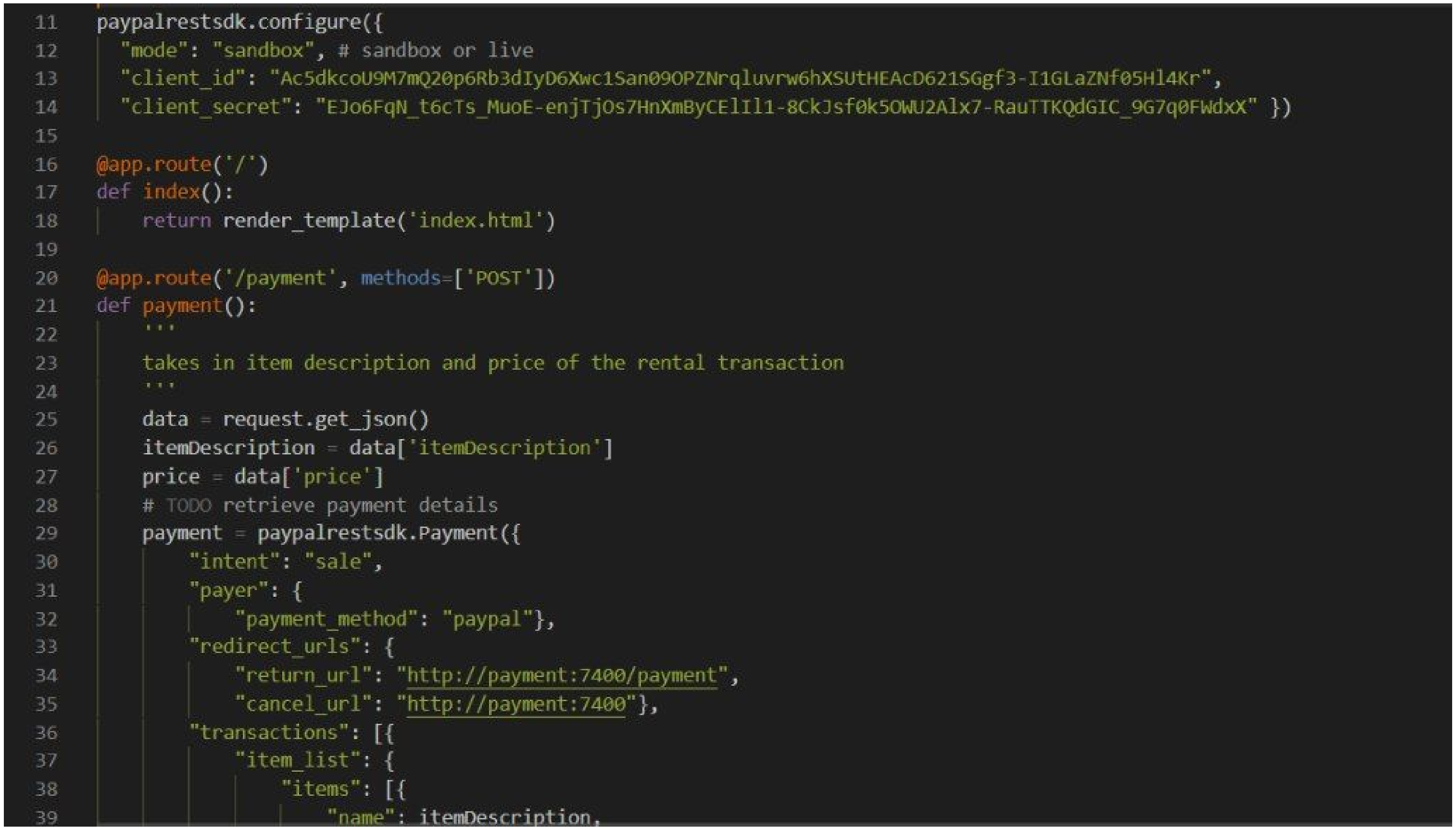
Honda 338

Daily Rate: 35

Color: Purple | Capacity: 5 | Tranmission Type: auto

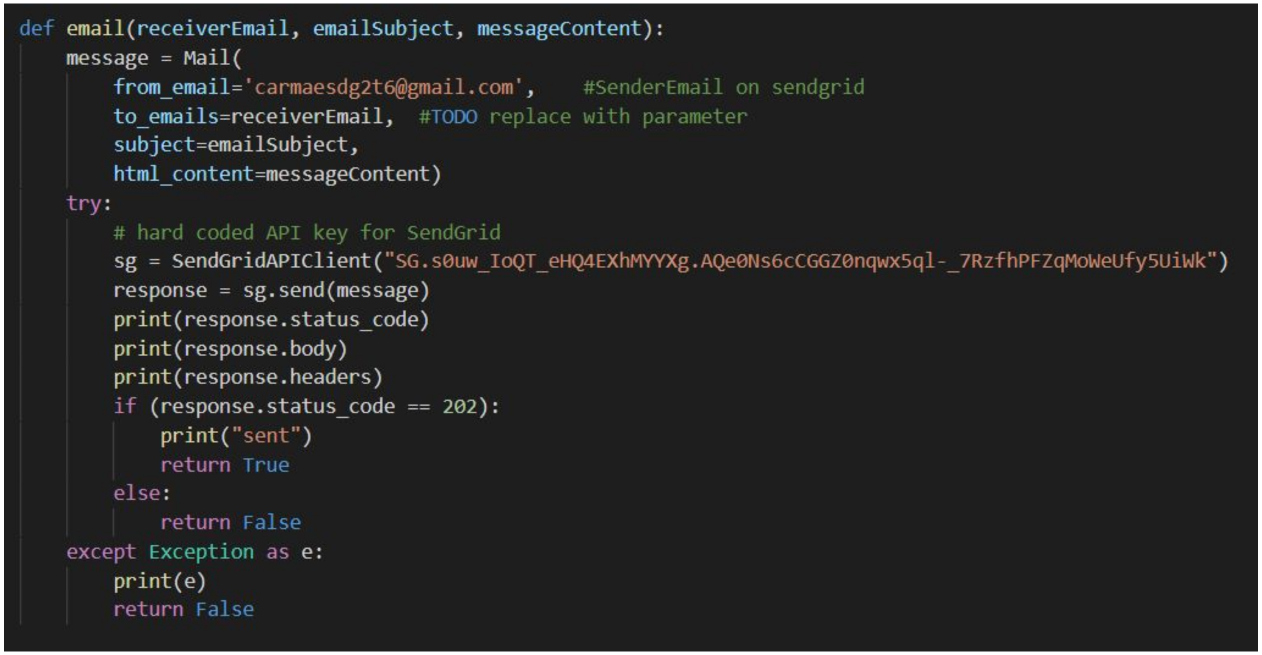
Distance from you: 11097m

PayPal API: refer to the submitted code for full version located in docker/payment/payment.py



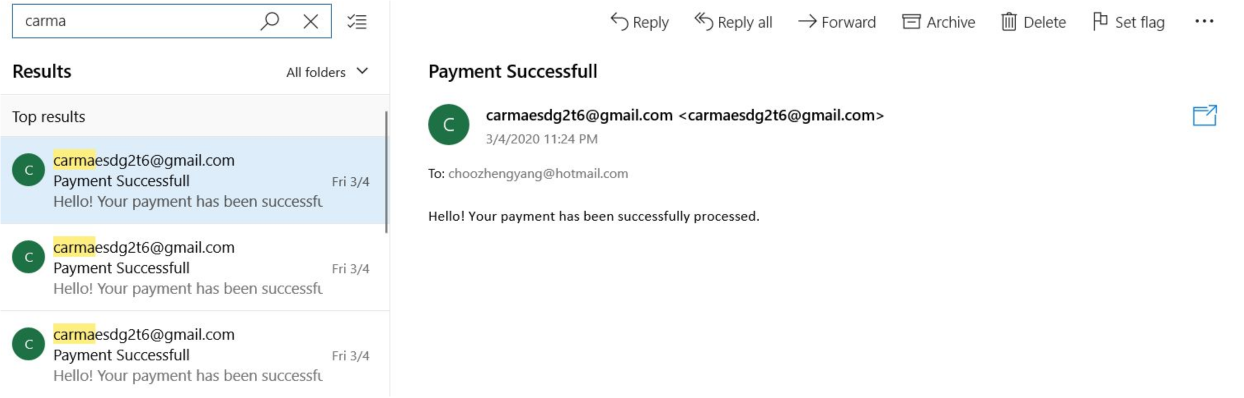
```
paypalrestsdk.configure({
  "mode": "sandbox", # sandbox or live
  "client_id": "Ac5dkcoU9M7mQ20p6Rb3dIyD6Xwc1San09OPZNrqluvrw6hXSUtHEAcD621SGgf3-I1GLaZNf05Hl4Kr",
  "client_secret": "EJo6FqN_t6cTs_MuoE-enjTjOs7HnXmByCElIl1-8CkJsf0k5OWU2Alx7-RauTTKQdGIC_9G7q0FWdxX" })

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/payment', methods=['POST'])
def payment():
    '''
    takes in item description and price of the rental transaction
    '''
    data = request.get_json()
    itemDescription = data['itemDescription']
    price = data['price']
    # TODO retrieve payment details
    payment = paypalrestsdk.Payment({
        "intent": "sale",
        "payer": {
            "payment_method": "paypal"},
        "redirect_urls": {
            "return_url": "http://payment:7400/payment",
            "cancel_url": "http://payment:7400"},
        "transactions": [{
            "item_list": {
                "items": [{
                    "name": itemDescription,
```

SendGrid API: refer to the submitted code for the codes we used to implement Telegram API in the notifications folder, in the email.py file.



```python
def email(receiverEmail, emailSubject, messageContent):
    message = Mail(
        from_email='carmaesdg2t6@gmail.com',    #SenderEmail on sendgrid
        to_emails=receiverEmail,  #TODO replace with parameter
        subject=emailSubject,
        html_content=messageContent)
    try:
        # hard coded API key for SendGrid
        sg = SendGridAPIClient("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")
        response = sg.send(message)
        print(response.status_code)
        print(response.body)
        print(response.headers)
        if (response.status_code == 202):
            print("sent")
            return True
        else:
            return False
    except Exception as e:
        print(e)
        return False
```

Evidence of SendGridAPI working as shown:



carmaesdg2t6@gmail.com
Payment Successfull
Hello! Your payment has been successfu
Fri 3/4

carmaesdg2t6@gmail.com
Payment Successfull
Hello! Your payment has been successfu
Fri 3/4

carmaesdg2t6@gmail.com
Payment Successfull
Hello! Your payment has been successfu
Fri 3/4

Payment Successfull

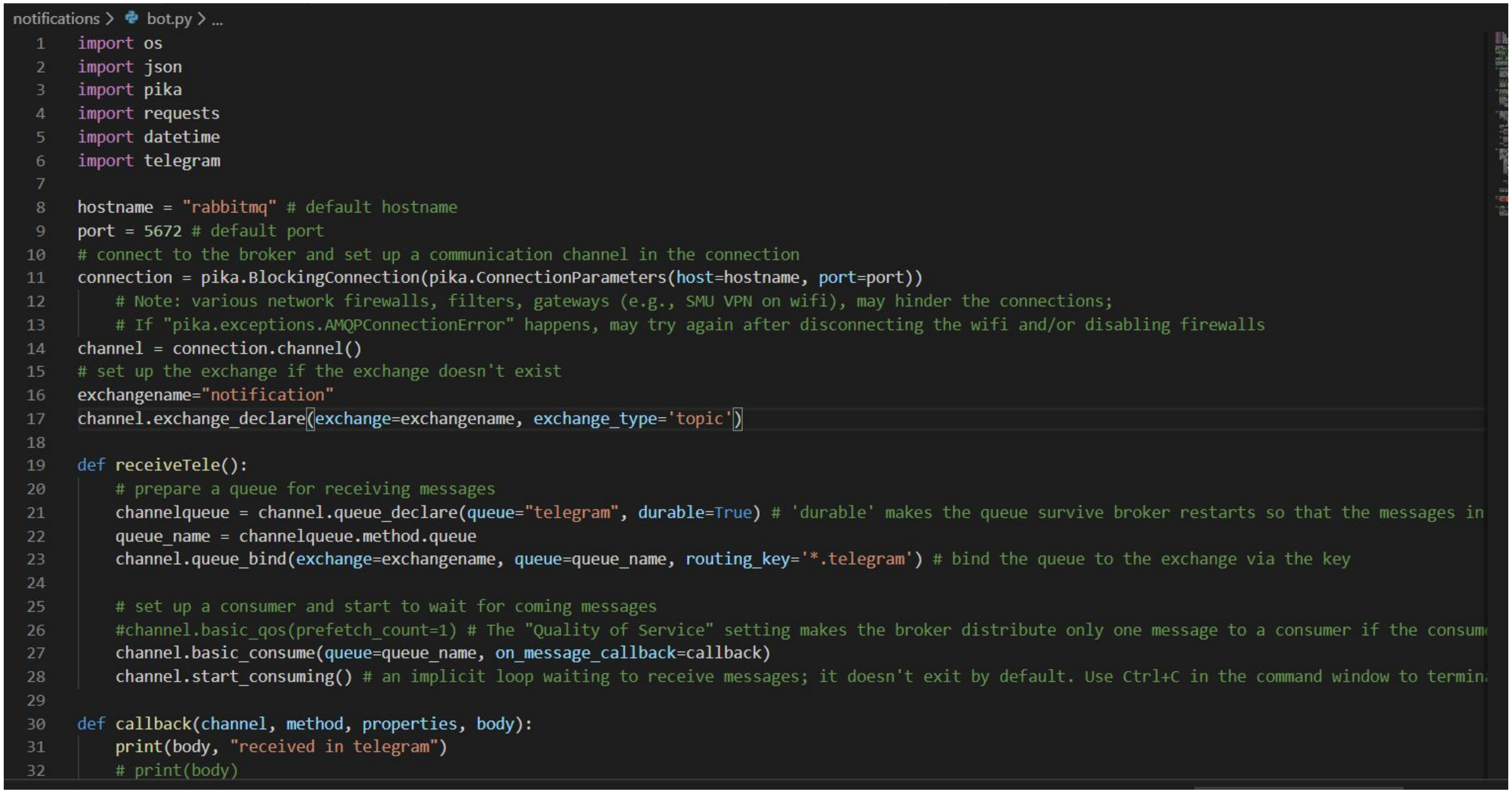
carmaesdg2t6@gmail.com <carmaesdg2t6@gmail.com>
3/4/2020 11:24 PM

To: choozhengyang@hotmail.com

Hello! Your payment has been successfully processed.

Wrapper Microservice: In order for the notification microservice to interact with the external APIs, we implemented wrapper microservices such as Email and Bot for each external APIs for the data exchange. The wrapper microservices can be found in the notifications folder.

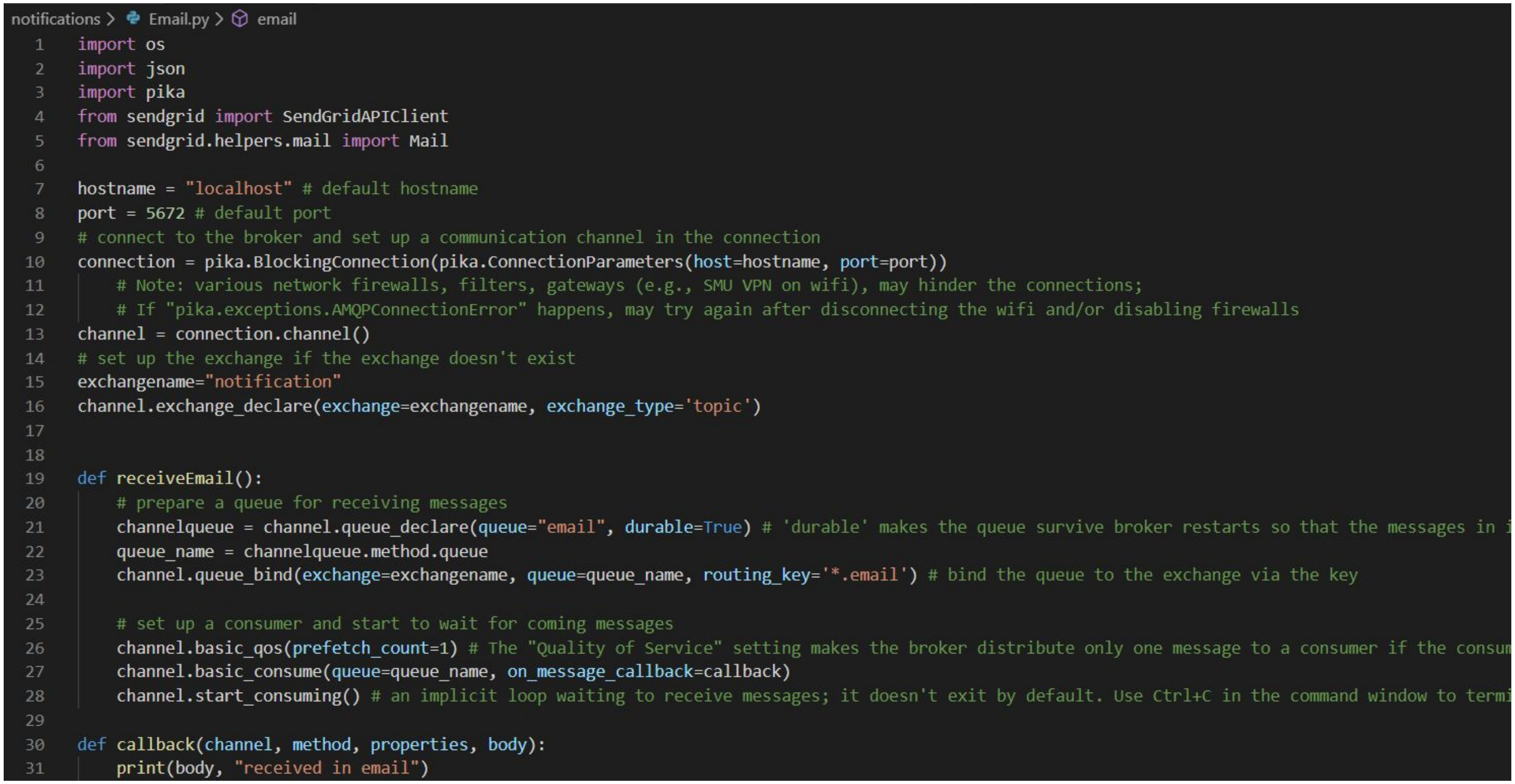
Telegram wrapper (bot.py):



notifications > bot.py > ...

```python
import os
import json
import pika
import requests
import datetime
import telegram

hostname = "rabbitmq" # default hostname
port = 5672 # default port
# connect to the broker and set up a communication channel in the connection
connection = pika.BlockingConnection(pika.ConnectionParameters(host=hostname, port=port))
    # Note: various network firewalls, filters, gateways (e.g., SMU VPN on wifi), may hinder the connections;
    # If "pika.exceptions.AMQPConnectionError" happens, may try again after disconnecting the wifi and/or disabling firewalls
channel = connection.channel()
# set up the exchange if the exchange doesn't exist
exchangename="notification"
channel.exchange_declare(exchange=exchangename, exchange_type='topic')

def receiveTele():
    # prepare a queue for receiving messages
    channelqueue = channel.queue_declare(queue="telegram", durable=True) # 'durable' makes the queue survive broker restarts so that the messages in
    queue_name = channelqueue.method.queue
    channel.queue_bind(exchange=exchangename, queue=queue_name, routing_key='*.telegram') # bind the queue to the exchange via the key

    # set up a consumer and start to wait for coming messages
    #channel.basic_qos(prefetch_count=1) # The "Quality of Service" setting makes the broker distribute only one message to a consumer if the consum
    channel.basic_consume(queue=queue_name, on_message_callback=callback)
    channel.start_consuming() # an implicit loop waiting to receive messages; it doesn't exit by default. Use Ctrl+C in the command window to termin

def callback(channel, method, properties, body):
    print(body, "received in telegram")
    # print(body)
```

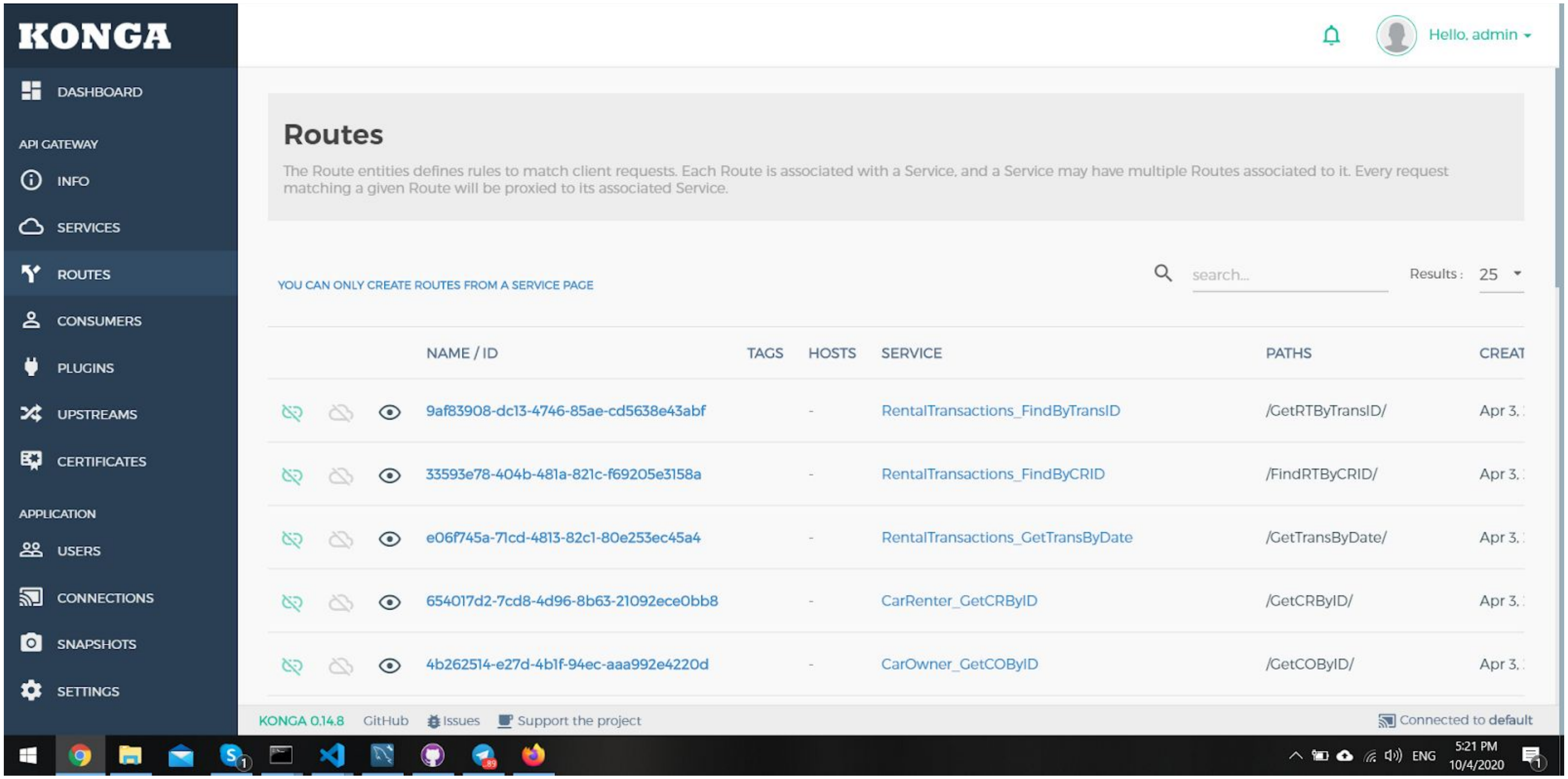
Email wrapper (Email.py):



```python
import os
import json
import pika
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail

hostname = "localhost" # default hostname
port = 5672 # default port
# connect to the broker and set up a communication channel in the connection
connection = pika.BlockingConnection(pika.ConnectionParameters(host=hostname, port=port))
    # Note: various network firewalls, filters, gateways (e.g., SMU VPN on wifi), may hinder the connections;
    # If "pika.exceptions.AMQPConnectionError" happens, may try again after disconnecting the wifi and/or disabling firewalls
channel = connection.channel()
# set up the exchange if the exchange doesn't exist
exchangename="notification"
channel.exchange_declare(exchange=exchangename, exchange_type='topic')


def receiveEmail():
    # prepare a queue for receiving messages
    channelqueue = channel.queue_declare(queue="email", durable=True) # 'durable' makes the queue survive broker restarts so that the messages in i
    queue_name = channelqueue.method.queue
    channel.queue_bind(exchange=exchangename, queue=queue_name, routing_key='*.email') # bind the queue to the exchange via the key

    # set up a consumer and start to wait for coming messages
    channel.basic_qos(prefetch_count=1) # The "Quality of Service" setting makes the broker distribute only one message to a consumer if the consum
    channel.basic_consume(queue=queue_name, on_message_callback=callback)
    channel.start_consuming() # an implicit loop waiting to receive messages; it doesn't exit by default. Use Ctrl+C in the command window to termi

def callback(channel, method, properties, body):
    print(body, "received in email")
```

Kong

The following shows a part of our routes set up:



Routes

The Route entities defines rules to match client requests. Each Route is associated with a Service, and a Service may have multiple Routes associated to it. Every request matching a given Route will be proxied to its associated Service.

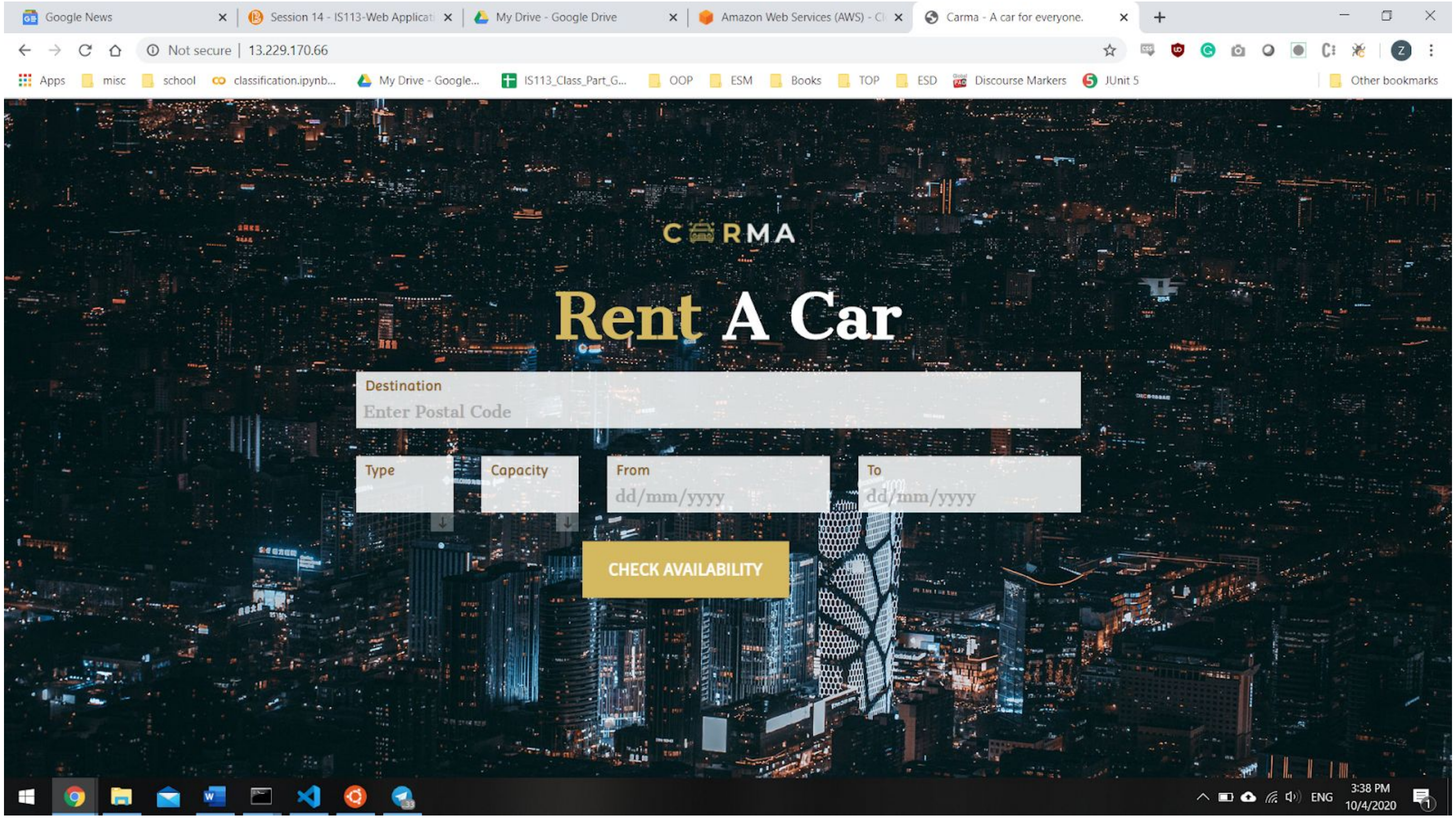
YOU CAN ONLY CREATE ROUTES FROM A SERVICE PAGE

NAME / ID	TAGS	HOSTS	SERVICE	PATHS	CREAT
9af83908-dc13-4746-85ae-cd5638e43abf		-	RentalTransactions_FindByTransID	/GetRTByTransID/	Apr 3,
33593e78-404b-481a-821c-f69205e3158a		-	RentalTransactions_FindByCRID	/FindRTByCRID/	Apr 3,
e06f745a-71cd-4813-82c1-80e253ec45a4		-	RentalTransactions_GetTransByDate	/GetTransByDate/	Apr 3,
654017d2-7cd8-4d96-8b63-21092ece0bb8		-	CarRenter_GetCRByID	/GetCRByID/	Apr 3,
4b262514-e27d-4b1f-94ec-aaa992e4220d		-	CarOwner_GetCOByID	/GetCOByID/	Apr 3,

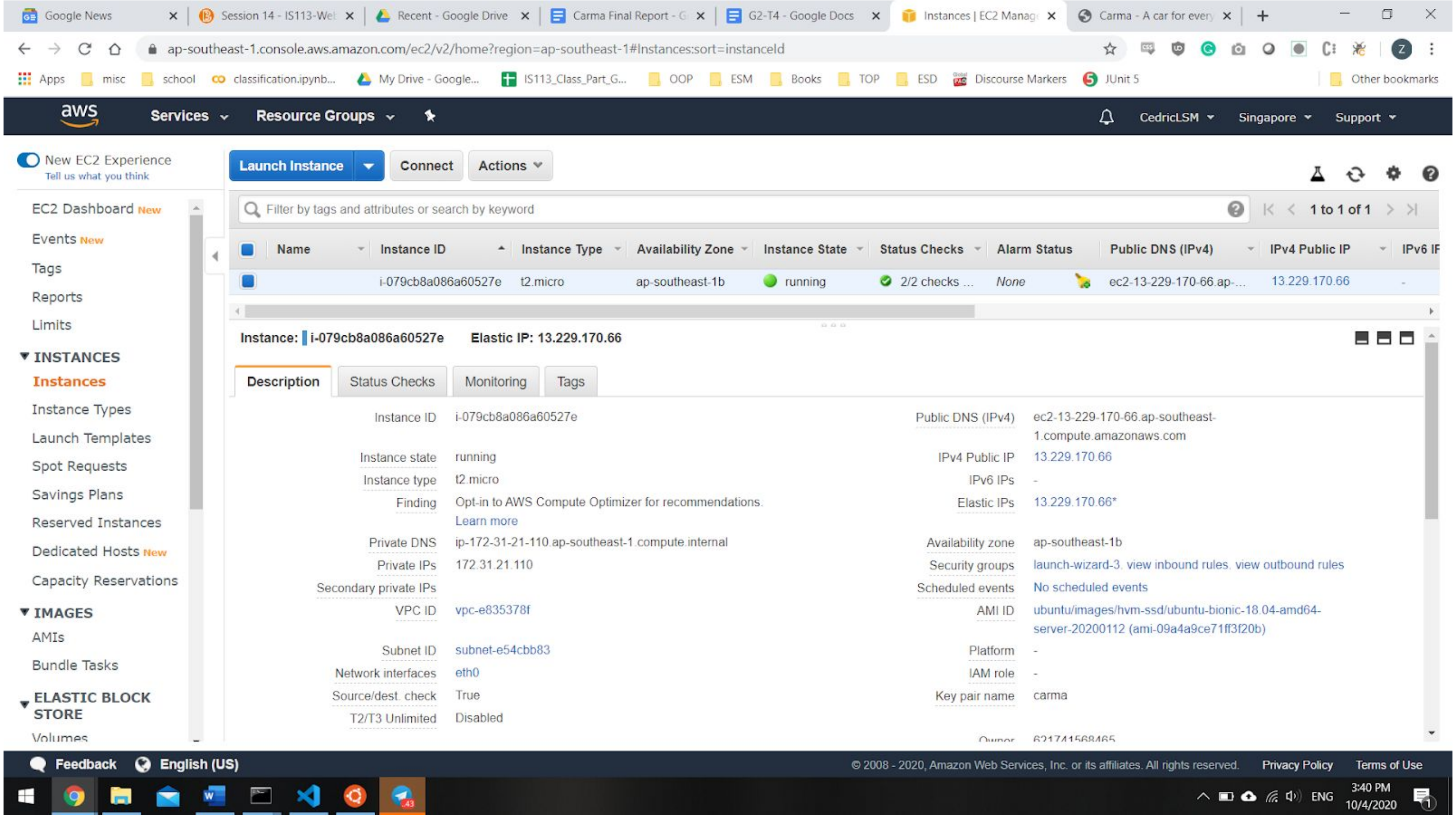
AWS Deployment

The team also deployed our microservice and front end on an AWS instance with an elastic IP 13.229.170.66. We deployed it by pushing our containers onto Docker Hub and SSH into the instance to pull and run the containers. We pulled the images using docker-compose which can be found in the docker-compose(pull).yml file in our submission.

However, we have shut down the instance after recording the demo due to costs. The AWS account is also running another instance for another project which we are required to keep running up to week 14. As we are on the free tier of AWS, we would incur costs if we keep the instance for ESD running. Instructor Eng Kit approved of it after we informed him.



Screenshot of AWS EC2 instance and Elastic IP:



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IP
	i-079cb8a086a60527e	t2.micro	ap-southeast-1b	● running	✓ 2/2 checks ...	None	ec2-13-229-170-66.ap-...	13.229.170.66	-

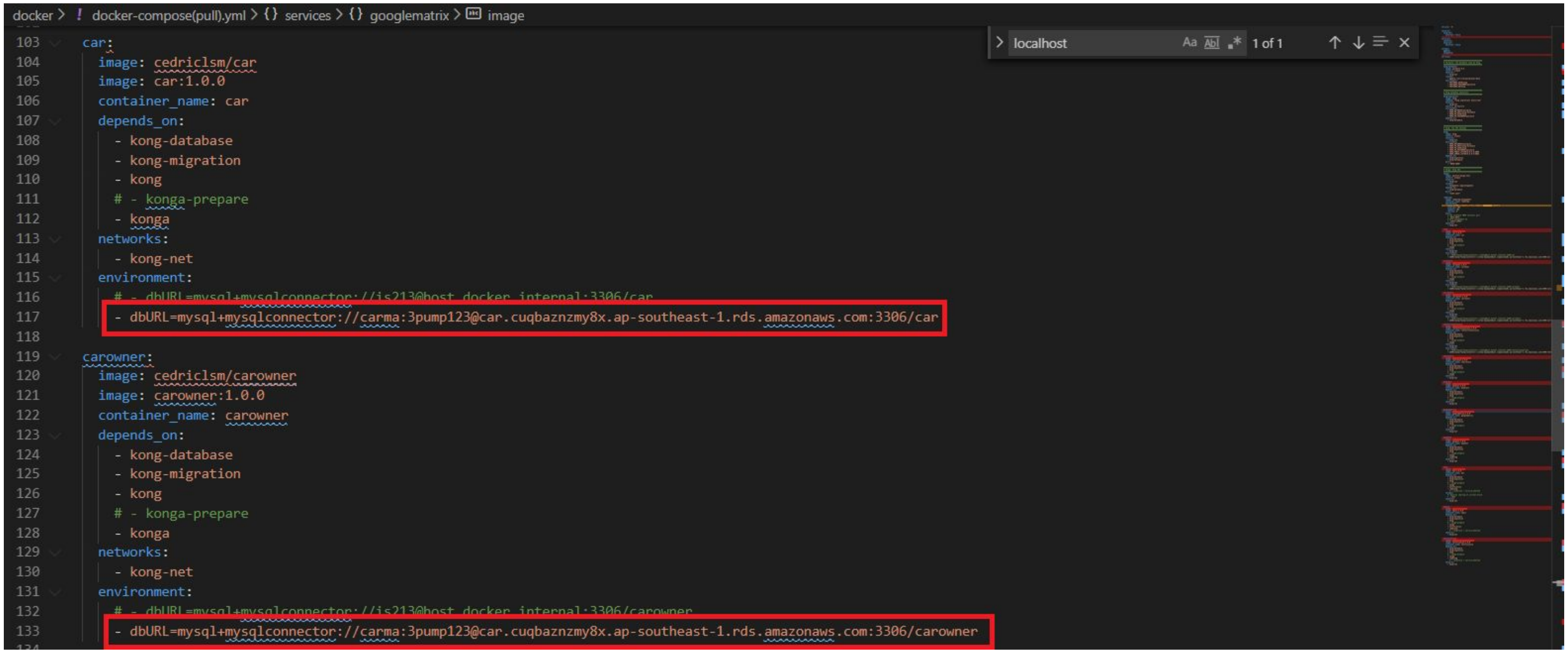
Instance: i-079cb8a086a60527e   Elastic IP: 13.229.170.66

Instance ID	i-079cb8a086a60527e	Public DNS (IPv4)	ec2-13-229-170-66.ap-southeast-1.compute.amazonaws.com
Instance state	running	IPv4 Public IP	13.229.170.66
Instance type	t2.micro	IPv6 IPs	-
Finding	Opt-in to AWS Compute Optimizer for recommendations. Learn more	Elastic IPs	13.229.170.66*
Private DNS	ip-172-31-21-110.ap-southeast-1.compute.internal	Availability zone	ap-southeast-1b
Private IPs	172.31.21.110	Security groups	launch-wizard-3. view inbound rules. view outbound rules
Secondary private IPs		Scheduled events	No scheduled events
VPC ID	vpc-e835378f	AMI ID	ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-20200112 (ami-09a4a9ce71ff3f20b)
Subnet ID	subnet-e54cbb83	Platform	-
Network interfaces	eth0	IAM role	-
Source/dest. check	True	Key pair name	carma
T2/T3 Unlimited	Disabled		
		Owner	621741568465

© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

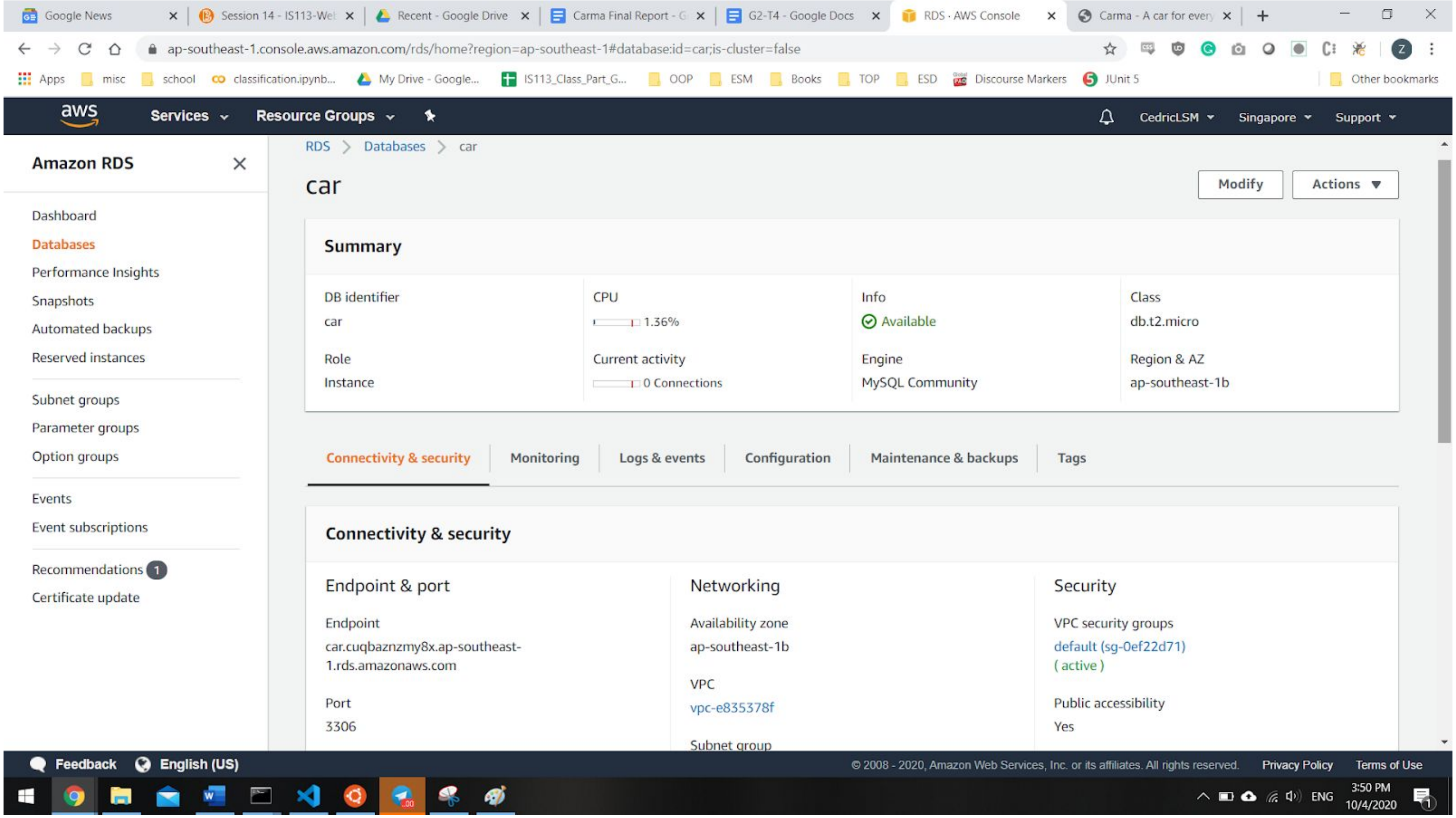
RDS Database

The team has also decoupled our microservice from its data storage by utilising Amazon RDS. We configured our microservice to connect to the RDS database by passing it in as an environmental variable in our docker-compose.


docker > ! docker-compose(pull).yml > {} services > {} googlematrix > image
localhost
Aa
1 of 1

```
  car:
    image: cedriclsm/car
    image: car:1.0.0
    container_name: car
    depends_on:
      - kong-database
      - kong-migration
      - kong
      # - konga-prepare
      - konga
    networks:
      - kong-net
    environment:
      # - dbURL=mysql+mysqlconnector://is213@host.docker.internal:3306/car
      - dbURL=mysql+mysqlconnector://carma:3pump123@car.cuqbaznzmy8x.ap-southeast-1.rds.amazonaws.com:3306/car

  carowner:
    image: cedriclsm/carowner
    image: carowner:1.0.0
    container_name: carowner
    depends_on:
      - kong-database
      - kong-migration
      - kong
      # - konga-prepare
      - konga
    networks:
      - kong-net
    environment:
      # - dbURL=mysql+mysqlconnector://is213@host.docker.internal:3306/carowner
      - dbURL=mysql+mysqlconnector://carma:3pump123@car.cuqbaznzmy8x.ap-southeast-1.rds.amazonaws.com:3306/carowner
```

Screenshot of RDS instance:



car

Summary

DB identifier	CPU	Info	Class
car	1.36%	✓ Available	db.t2.micro
Role	Current activity	Engine	Region & AZ
Instance	0 Connections	MySQL Community	ap-southeast-1b

Connectivity & security

Endpoint & port

Endpoint
car.cuqbaznzmy8x.ap-southeast-1.rds.amazonaws.com

Port
3306

Networking

Availability zone
ap-southeast-1b

VPC
vpc-e835378f

Subnet group

Security

VPC security groups
default (sg-0ef22d71)
(active)

Public accessibility
Yes