Shopping Website

Team: Cloud Players
Juntian Sun: js5940
Zijian Zhang: zz2994
Yi Zhang: yz4339
Yifan Wang: yw3914
Lihui Yan: ly2593

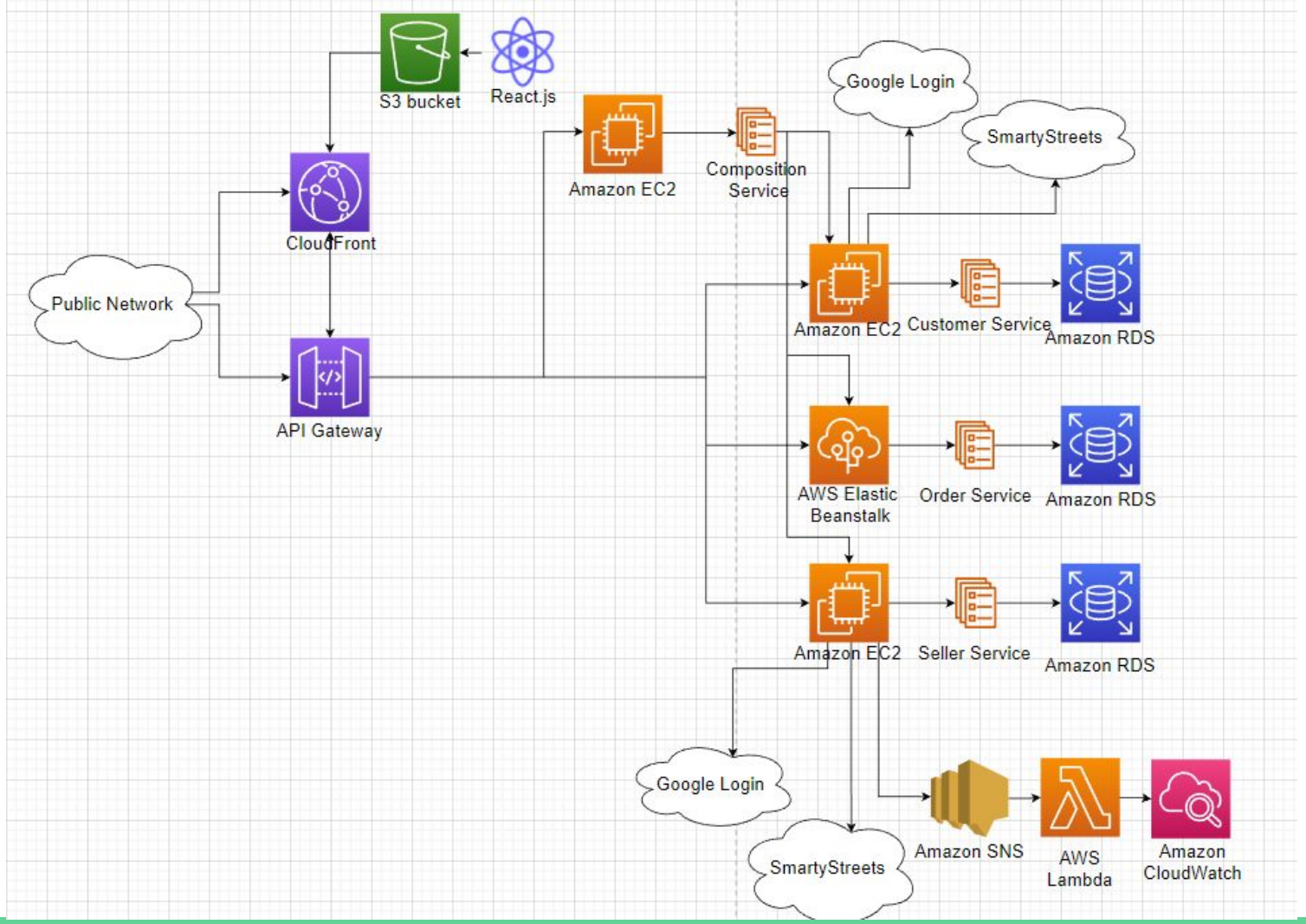
UI demo

Github for frontend https://github.com/sqwy/FullStackReact.git

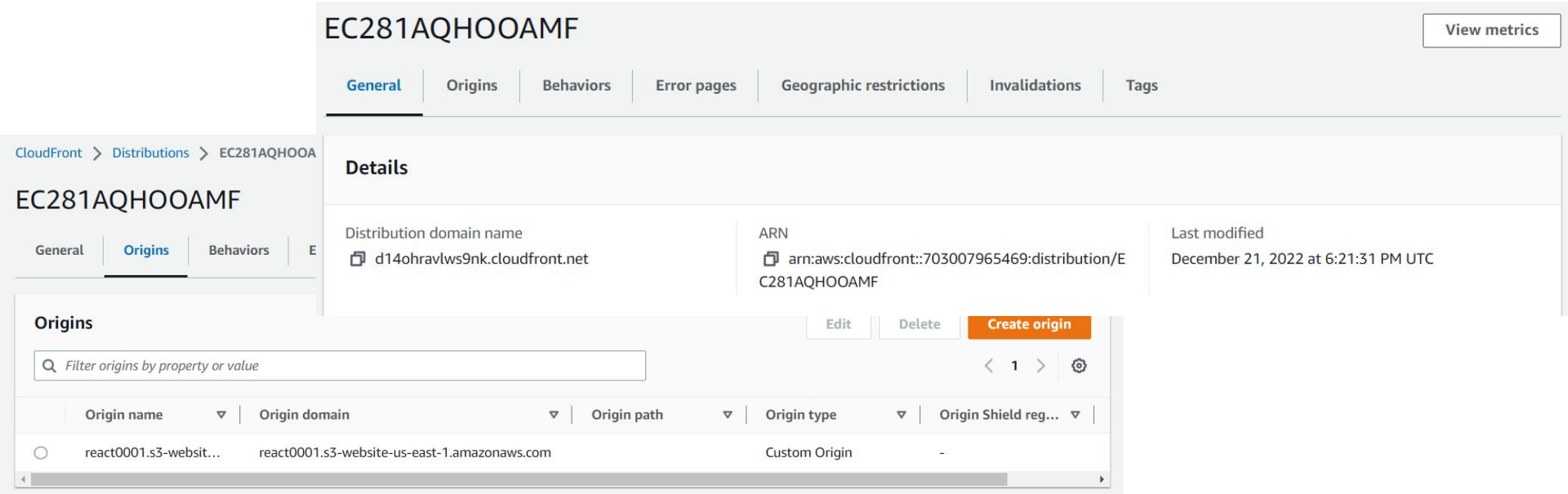
Our project

- Customer Service
- Seller Service
- Order service
- Composite Service

Our Project: Architecture

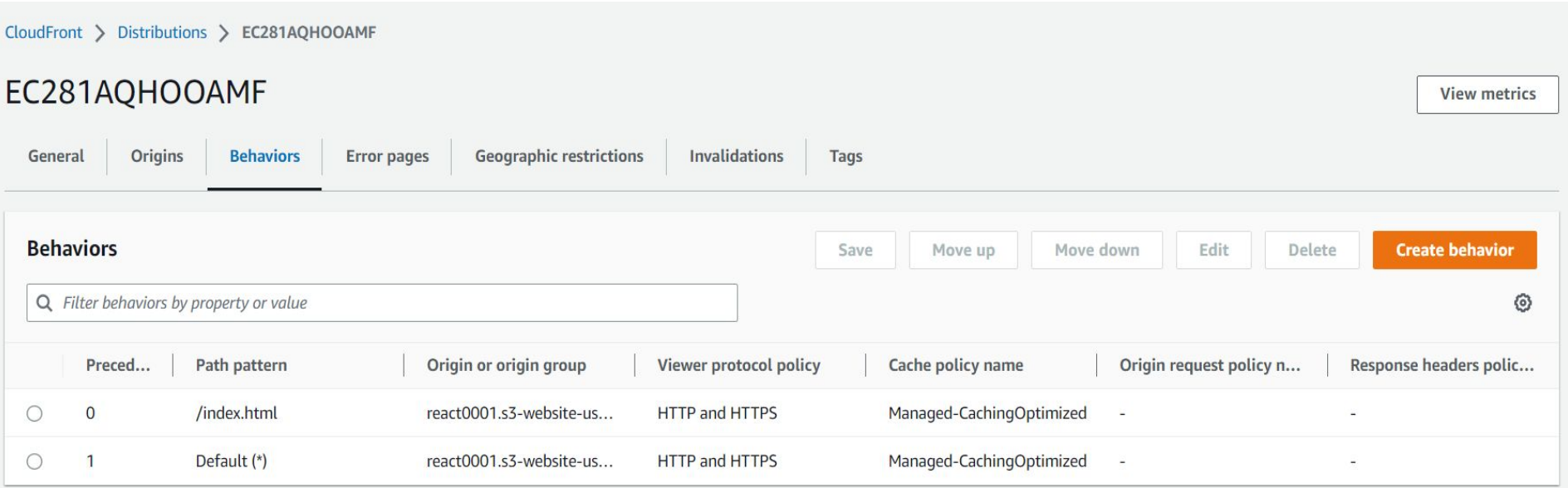


CloudFront

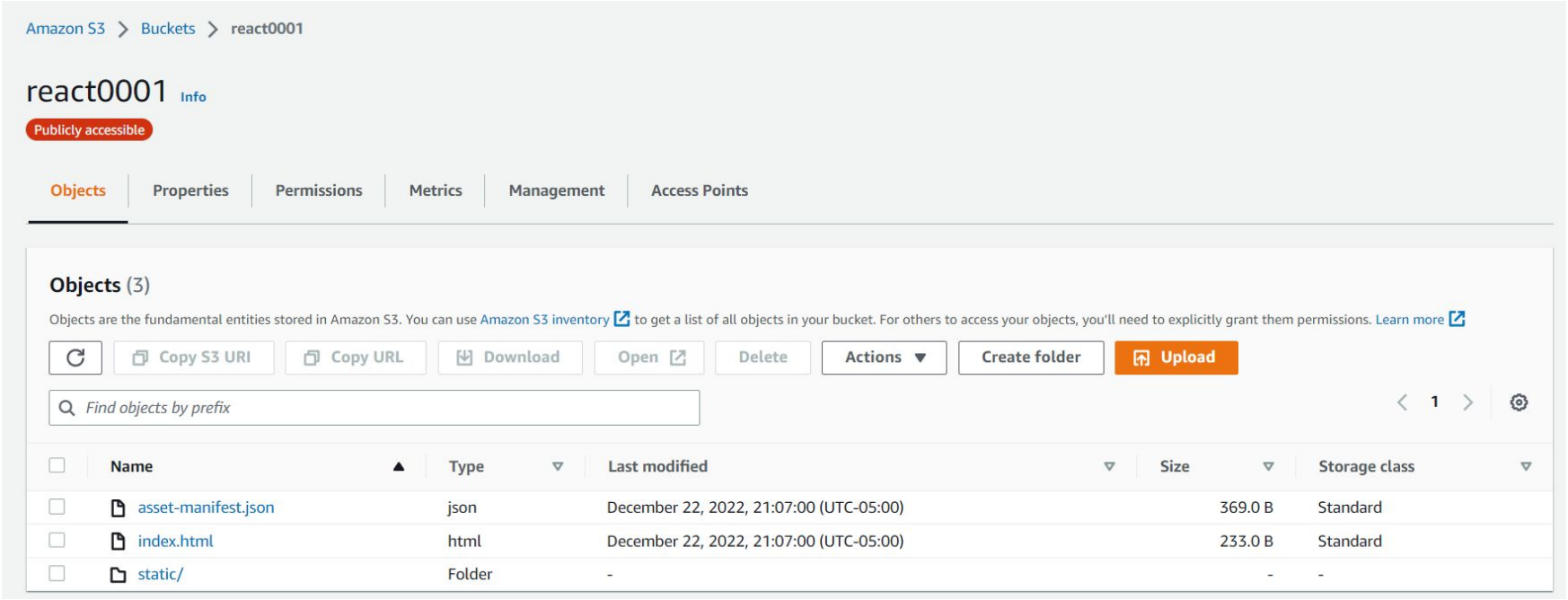


Link for cloudfront: https://d14ohravlws9nk.cloudfront.net

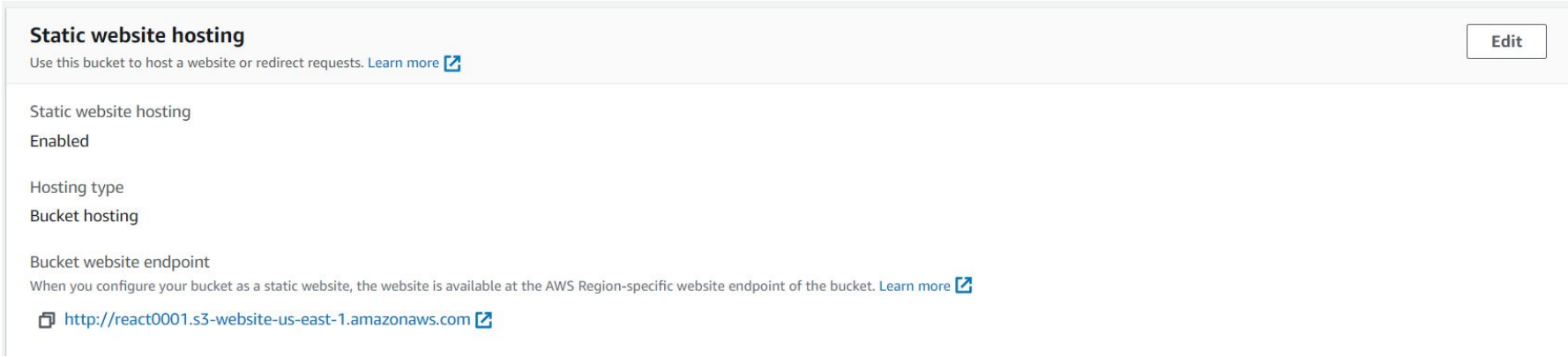
CloudFront Behaviors



S3 bucket

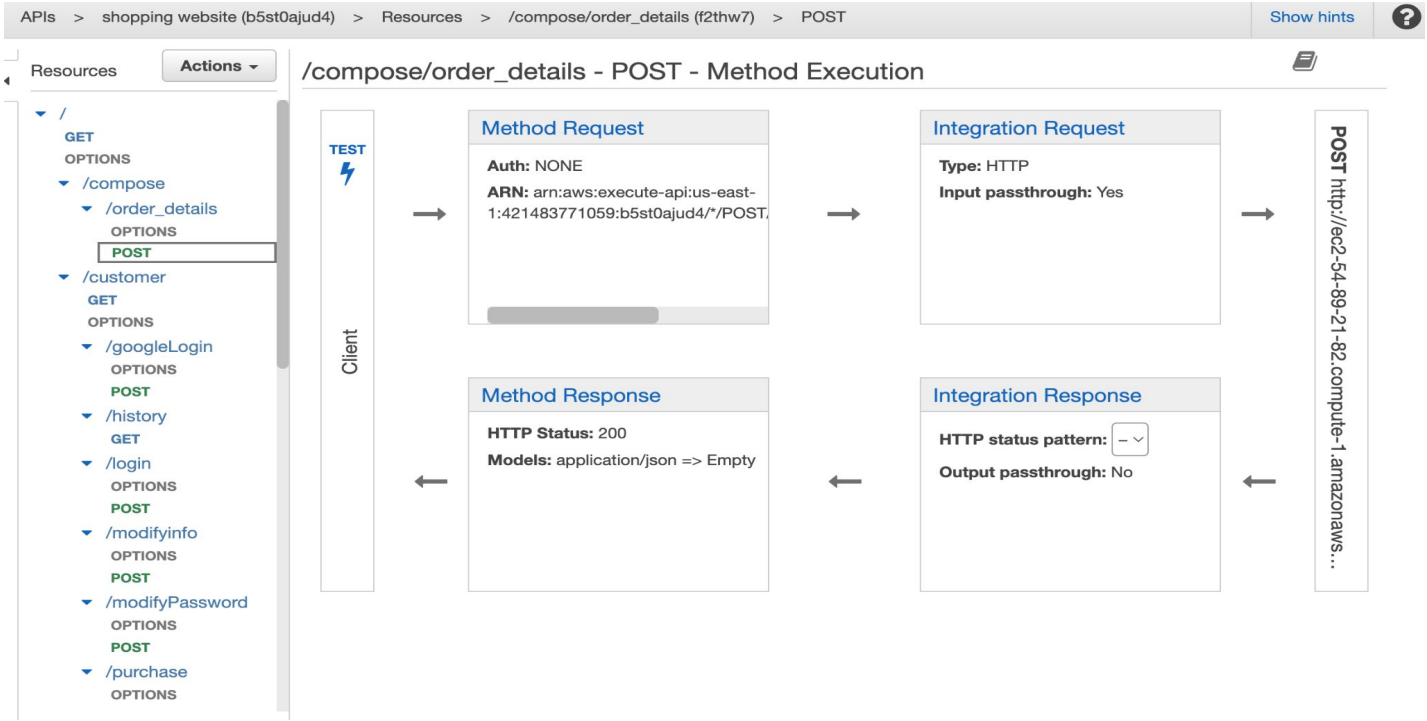

Amazon S3 > Buckets > react0001
react0001 Info
Publicly accessible
Objects
Properties
Permissions
Metrics
Management
Access Points
Objects (3)
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
1
Name
Type
Last modified
Size
Storage class
asset-manifest.json
json
December 22, 2022, 21:07:00 (UTC-05:00)
369.0 B
Standard
index.html
html
December 22, 2022, 21:07:00 (UTC-05:00)
233.0 B
Standard
static/
Folder
-
-
-

S3 static website hosting



Link for S3: http://react0001.s3-website-us-east-1.amazonaws.com

API Gateway



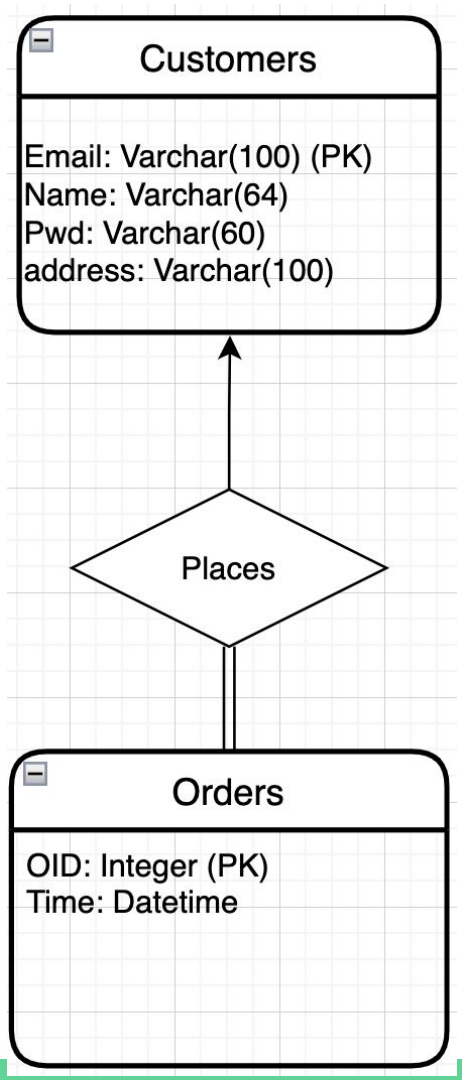
Link for API Gateway https://b5st0ajud4.execute-api.us-east-1.amazonaws.com/shopping

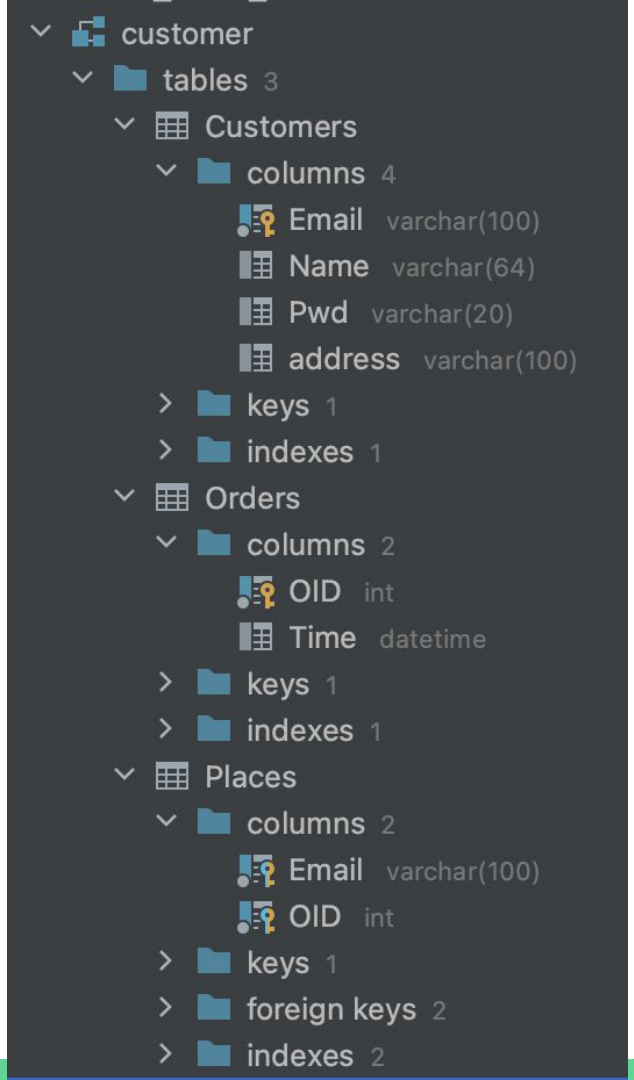
Customer service

Customer Service Overview

- Accept and respond with JSON
- Google login
- Connect a cloud database and browser application
- Interact with cloud database
- Smarty.com to validate and correct mailing addresses
- Middleware to check before signing in and after registering successfully
- Notification with SNS and Lambda functions after registering
- Allows pagination

Customer Service ER Diagram And Schema

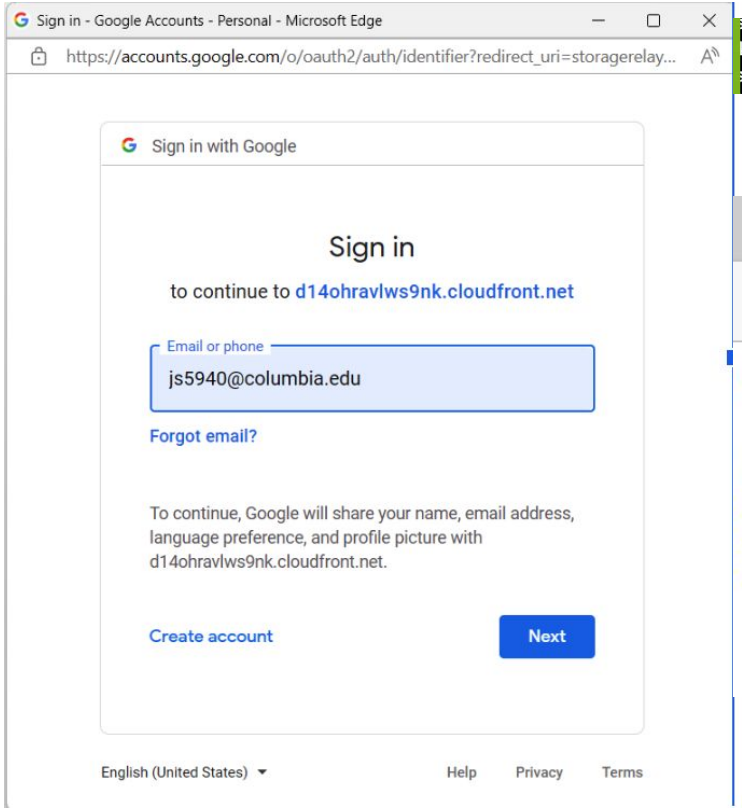


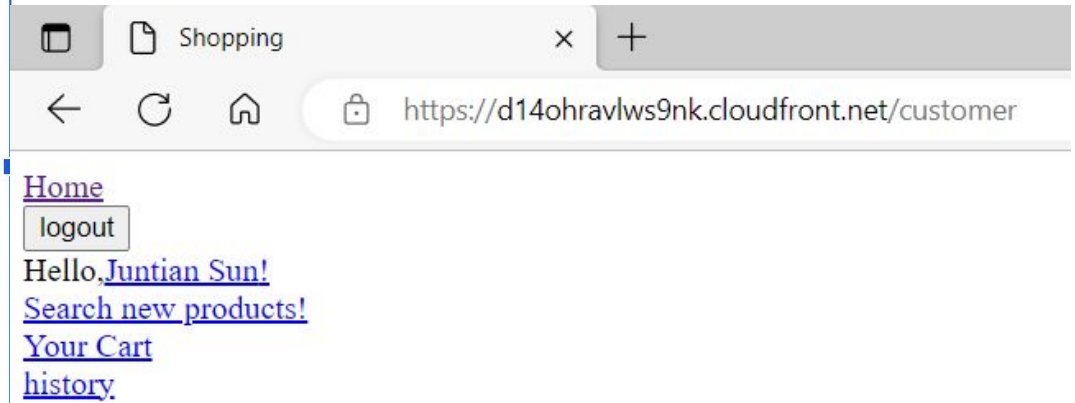


Github for Customer

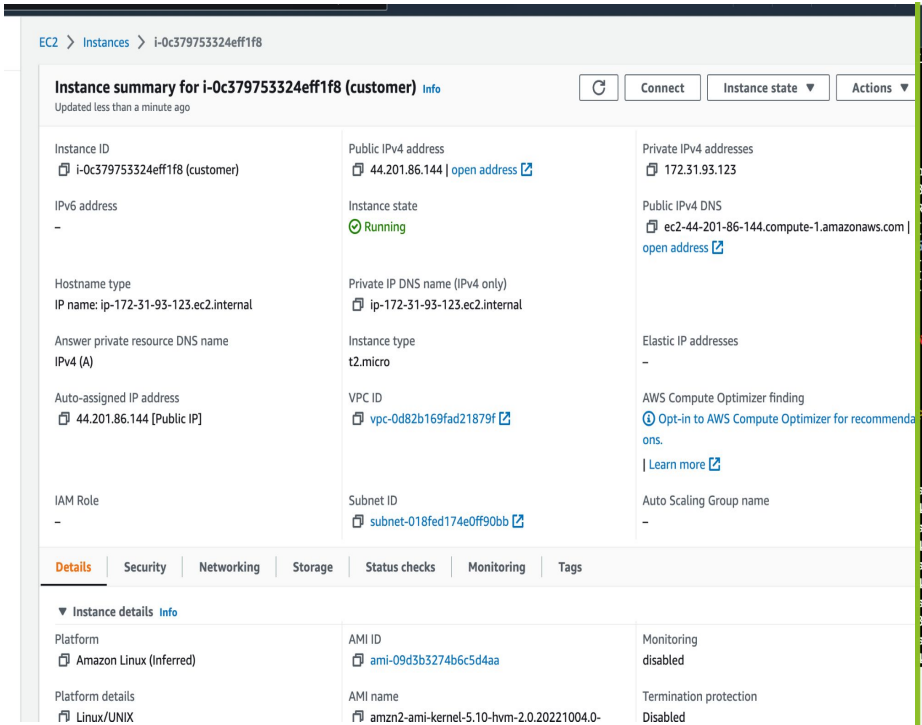
https://github.com/ywang289/customer.git

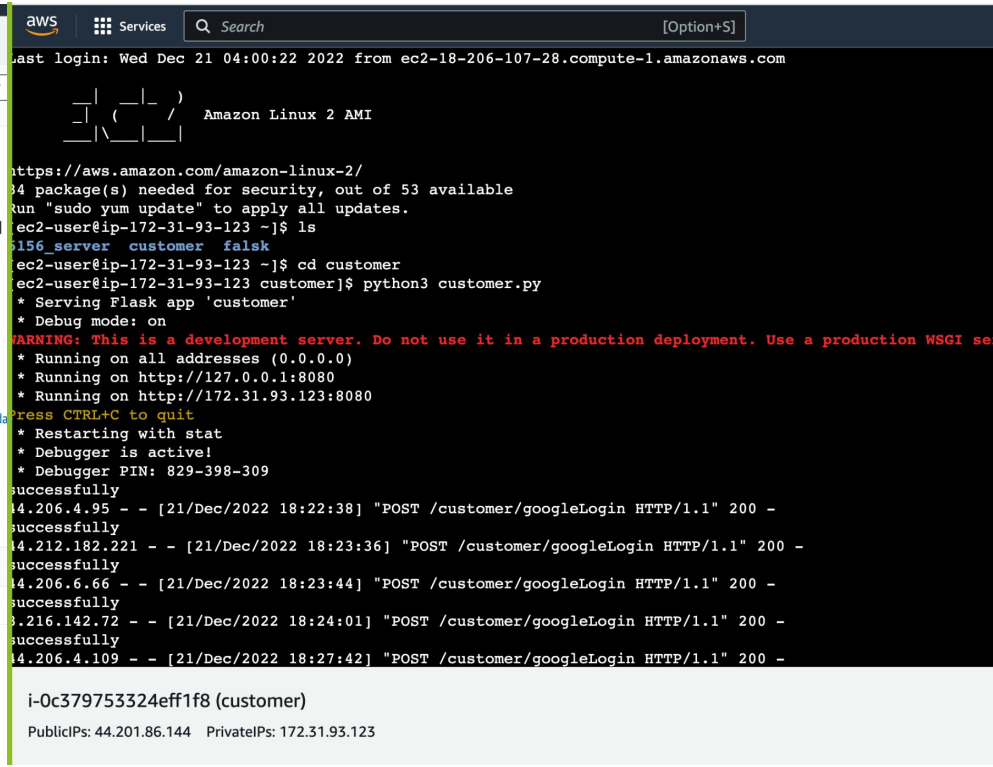
Google Login Oauth





EC2





Customer service: https://ec2-44-201-86-144.compute-1.amazonaws.com

Middleware

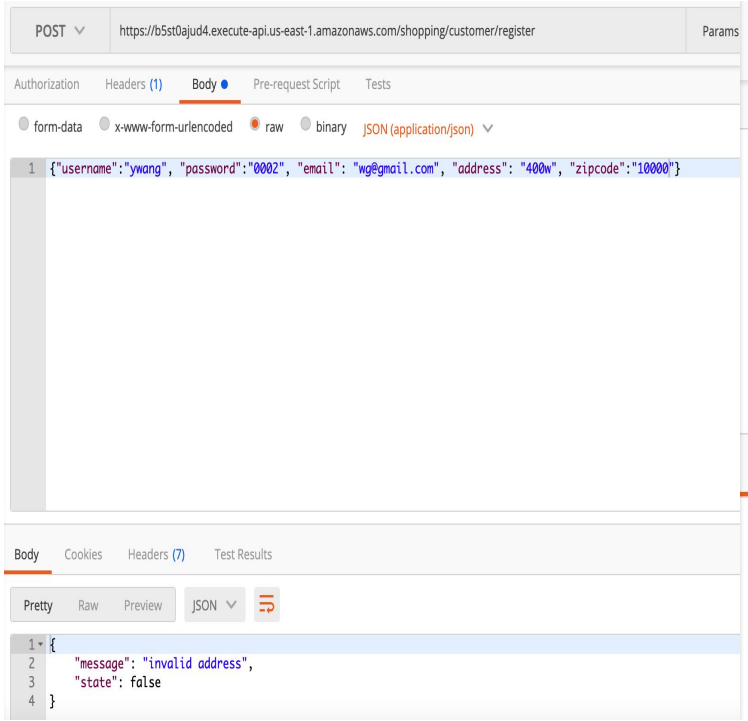
- Check email and password before logging in
- Check address before registration
- Send email to manager after user registration successfully

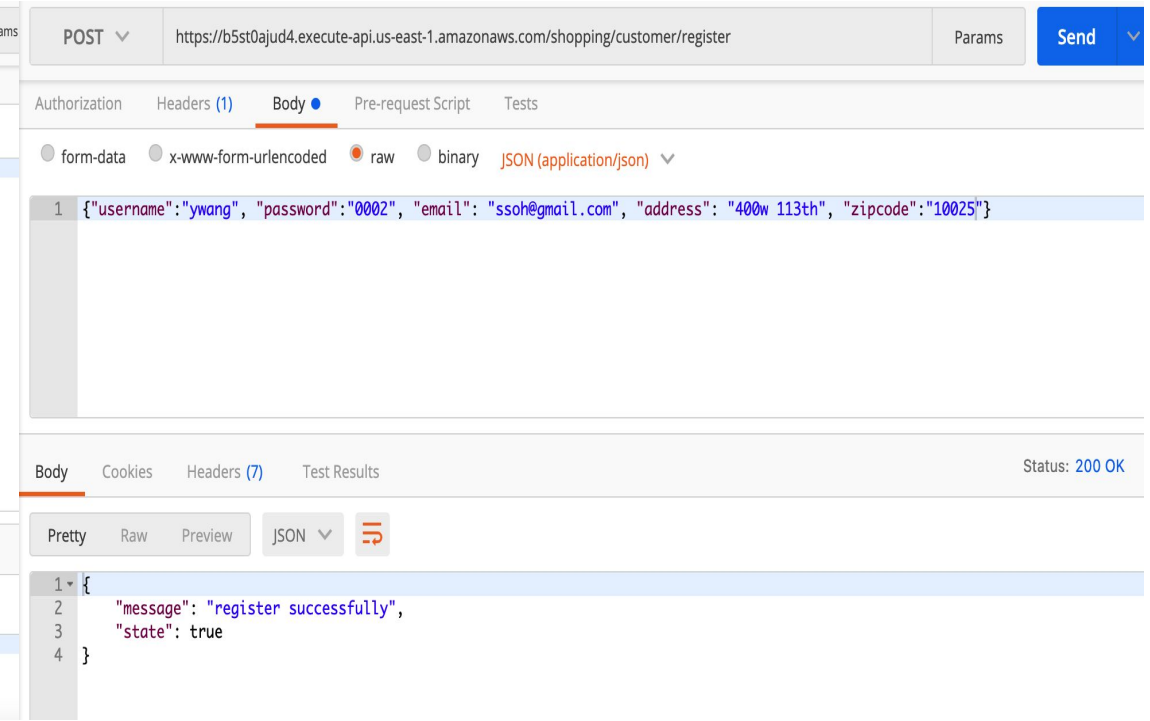
```python
@app.before_request
def run():
    if request.path == '/customer/register':
        auth_id = "c832f79c-cc29-3a15-c118-51733a6b5929"
        auth_token = "CoGbY7DbR8qNdUPu3aZh"
        data = json.loads(request.get_data())      You, 1小时前 • Uncommitted changes
        address = data['address']
        zipcode= data["zipcode"]

        credentials = StaticCredentials(auth_id, auth_token)

        client = ClientBuilder(credentials).build_us_street_api_client()

        lookup = Lookup()

        lookup.street = address
        lookup.zipcode = zipcode


        try:
            client.send_lookup(lookup)
        except exceptions.SmartyException as err:
            print(err)
            return {"message": "input error", "state": False}

        result = lookup.result

        if not result:
            print("No candidates. This means the address is not valid.")
            return {"message": "invalid address", "state": False}

@app.after_request
def af3(response):
    if request.path == '/customer/register':
    # print(request.environ.get('SERVER_PROTOCOL'))
    # print(len((response.data.decode('utf-8').split(":"))))
    # print(json.loads(response.get_data())["state"])
```

Login Authentication

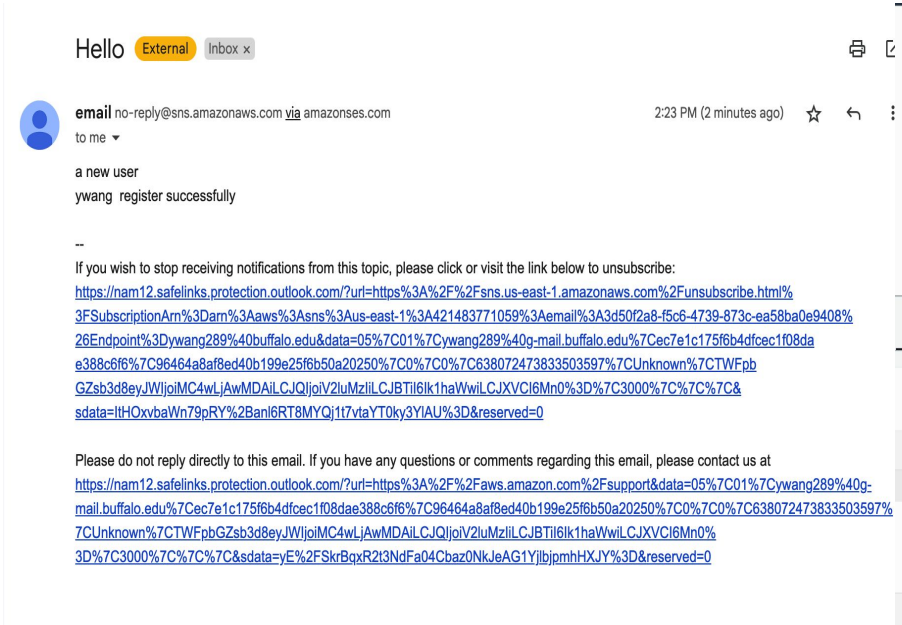
```
@app.before_request
def check_login():
    if request.path == '/seller/login':
        data = json.loads(request.get_data())
        password = data['password']
        email = data['email']

        try:
            sql = "SELECT * FROM Sellers where email = '{}' ".format(email)
            result = db.session.execute(sql).fetchone()
        except Exception as err:
            return {"state": False, "message": "error! input error"}
        if result:
            stored_password= result[2]
            if stored_password != password:

                print("unmatch")
                return {"state": False,"message":"password unmatch"}

        else:
            return {"state": False, "message": "please register"}
```

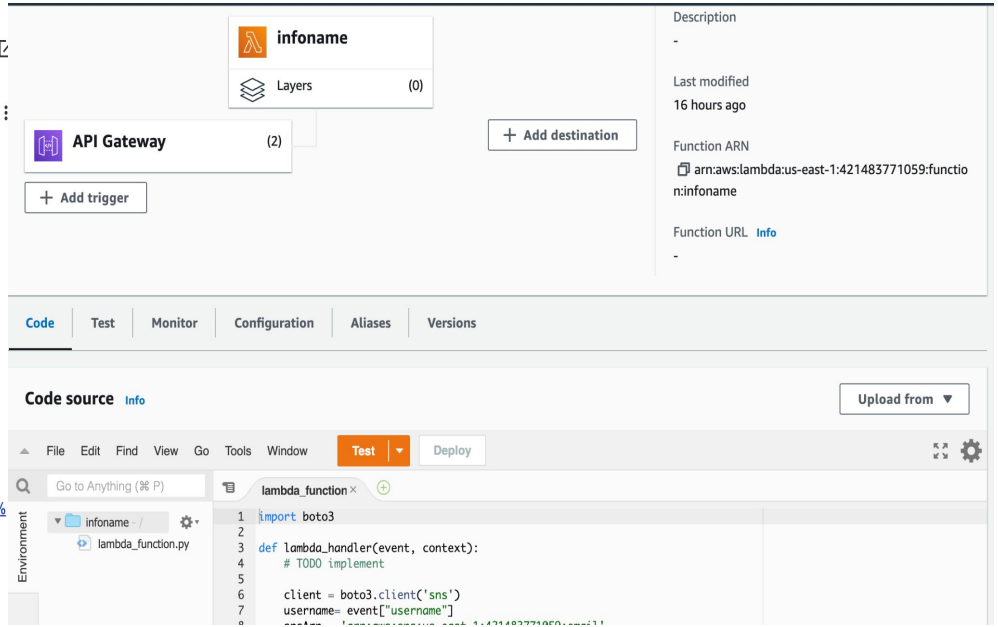
Customer service: Address Verification with SmartyStreets



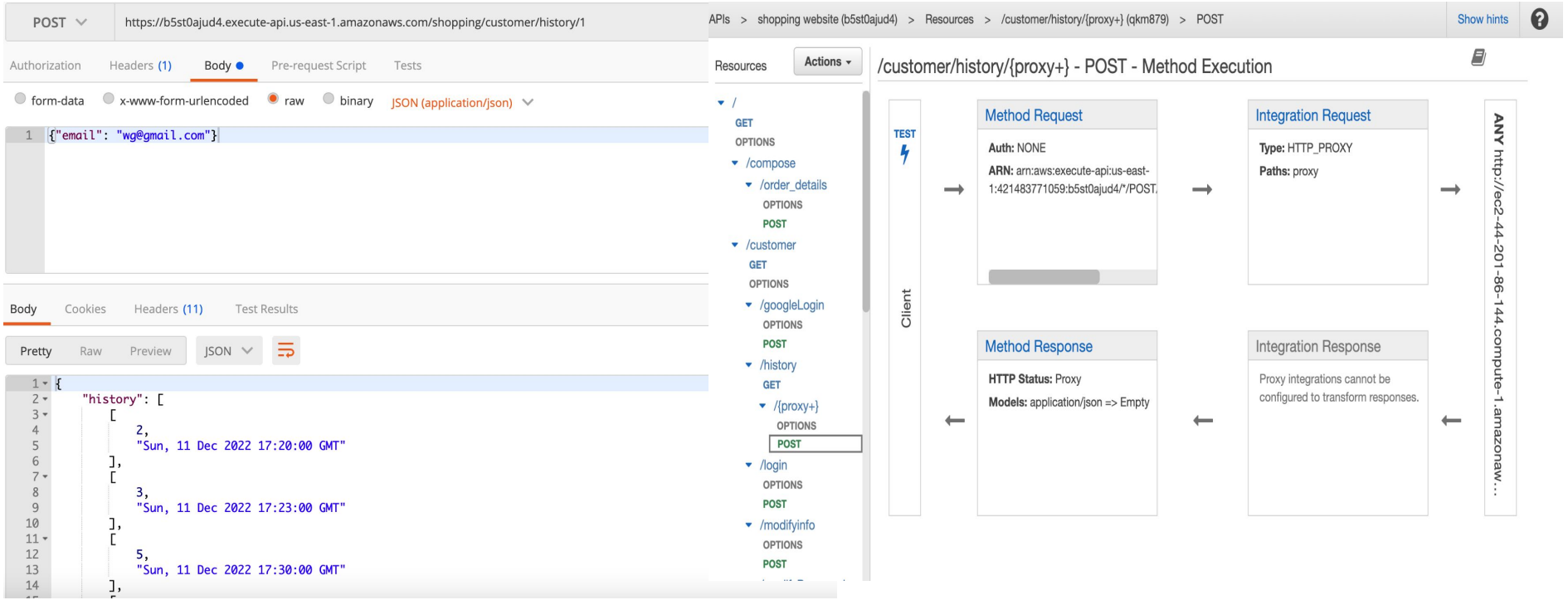


Customer service:
Notification with SNS and Lambda functions

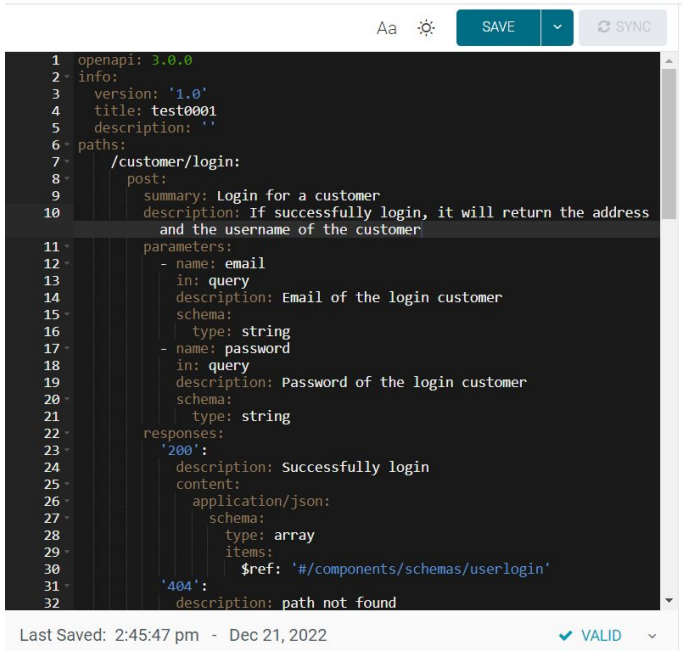


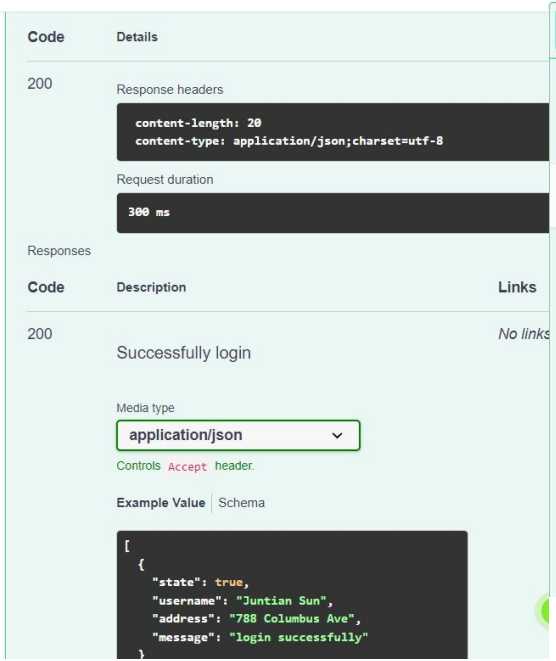


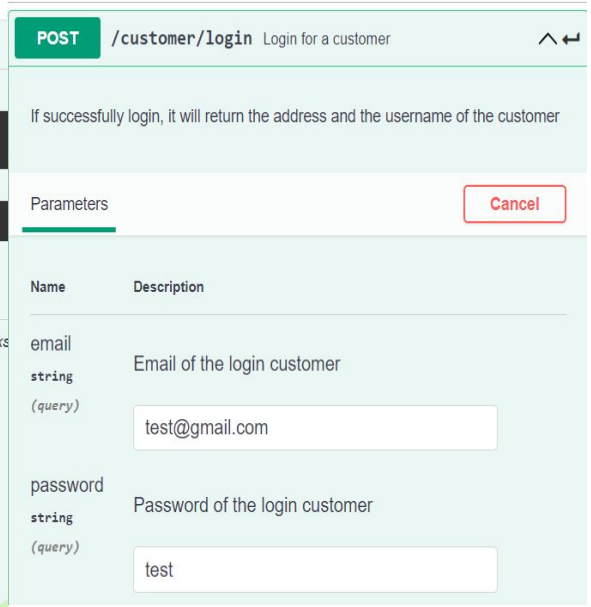
Pagination


POST
https://b5st0ajud4.execute-api.us-east-1.amazonaws.com/shopping/customer/history/1
Authorization
Headers (1)
Body
Pre-request Script
Tests
form-data
x-www-form-urlencoded
raw
binary
JSON (application/json)
{"email": "wg@gmail.com"}
Body
Cookies
Headers (11)
Test Results
Pretty
Raw
Preview
JSON
{
"history": [
[
2,
"Sun, 11 Dec 2022 17:20:00 GMT"
],
[
3,
"Sun, 11 Dec 2022 17:23:00 GMT"
],
[
5,
"Sun, 11 Dec 2022 17:30:00 GMT"
],
APIs > shopping website (b5st0ajud4) > Resources > /customer/history/{proxy+} (qkm879) > POST
Show hints
Resources
Actions
/
GET
OPTIONS
/compose
/order_details
OPTIONS
POST
/customer
GET
OPTIONS
/googleLogin
OPTIONS
POST
/history
GET
/{proxy+}
OPTIONS
POST
/login
OPTIONS
POST
/modifyinfo
OPTIONS
POST
/customer/history/{proxy+} - POST - Method Execution
TEST
Client
Method Request
Auth: NONE
ARN: arn:aws:execute-api:us-east-1:421483771059:b5st0ajud4/*/POST/
Integration Request
Type: HTTP_PROXY
Paths: proxy
ANY http://ec2-44-201-86-144.compute-1.amazonaw...
Method Response
HTTP Status: Proxy
Models: application/json => Empty
Integration Response
Proxy integrations cannot be configured to transform responses.

Customer Service: SwaggerHub Verified



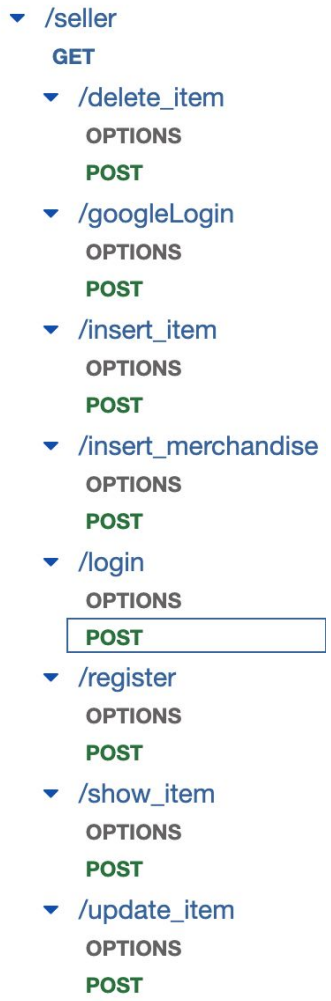




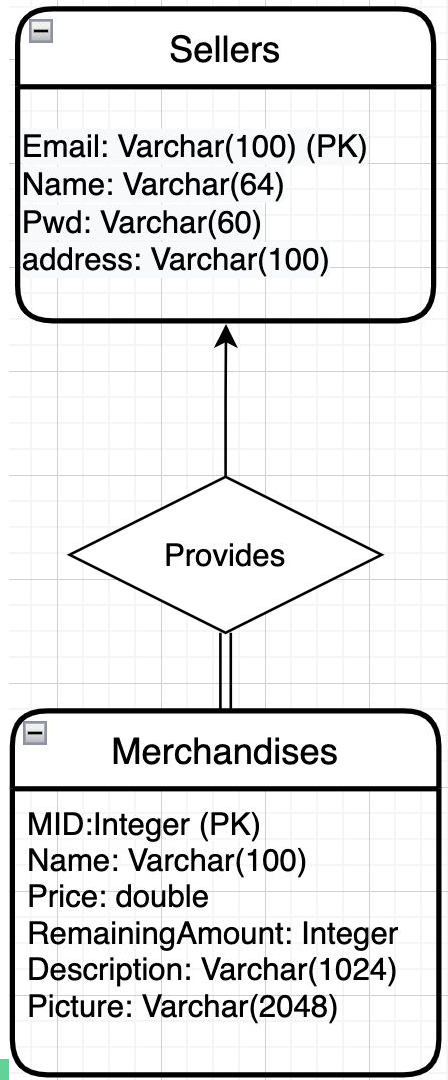
Seller service

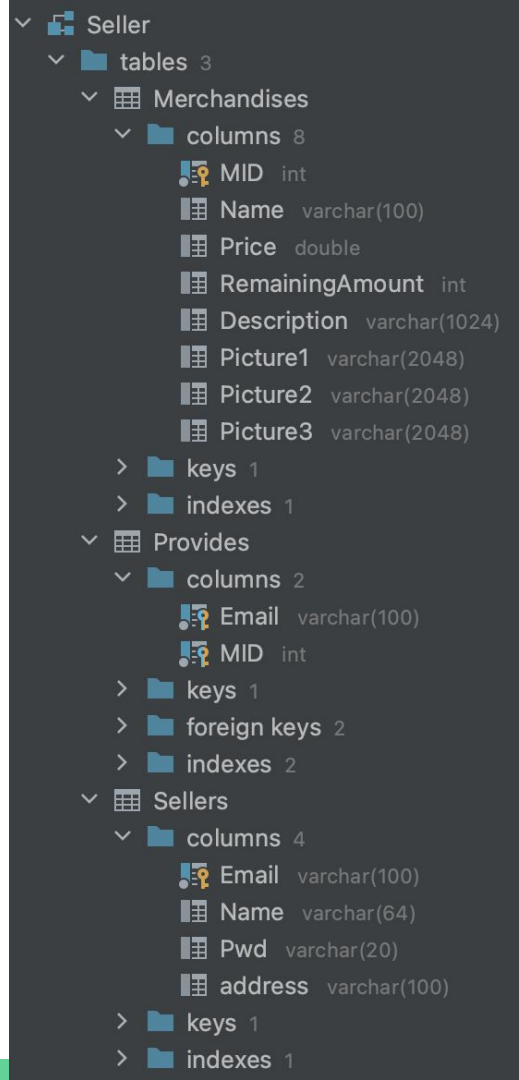
Seller Service Overview

- Accept and respond with JSON
- Google login
- Connect a cloud database and browser application
- Interact with cloud database
- Smarty.com to validate and correct mailing addresses
- Middleware to check before signing in and after registering successfully
- Notification with SNS and Lambda functions after registering
- Allows pagination



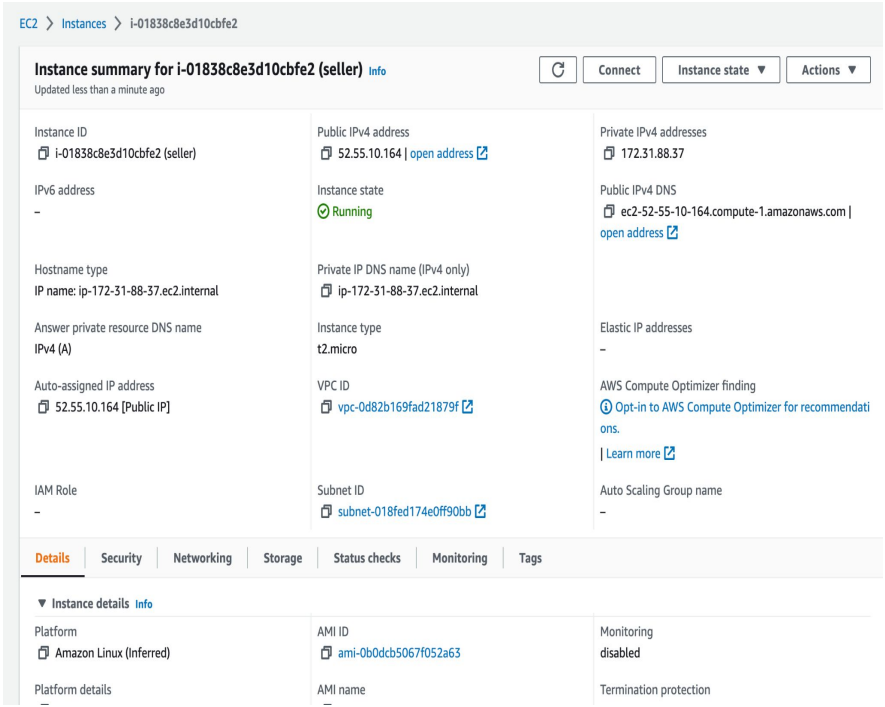
Seller Service ER Diagram And Schema

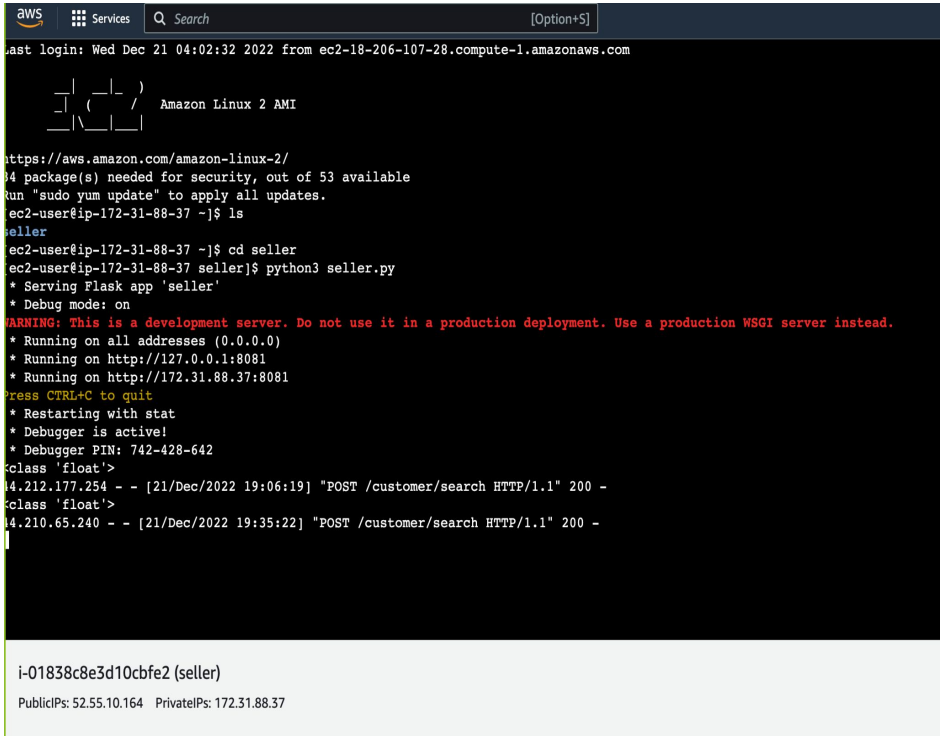




Github for Seller
https://github.com/ywang289/seller.git

EC2

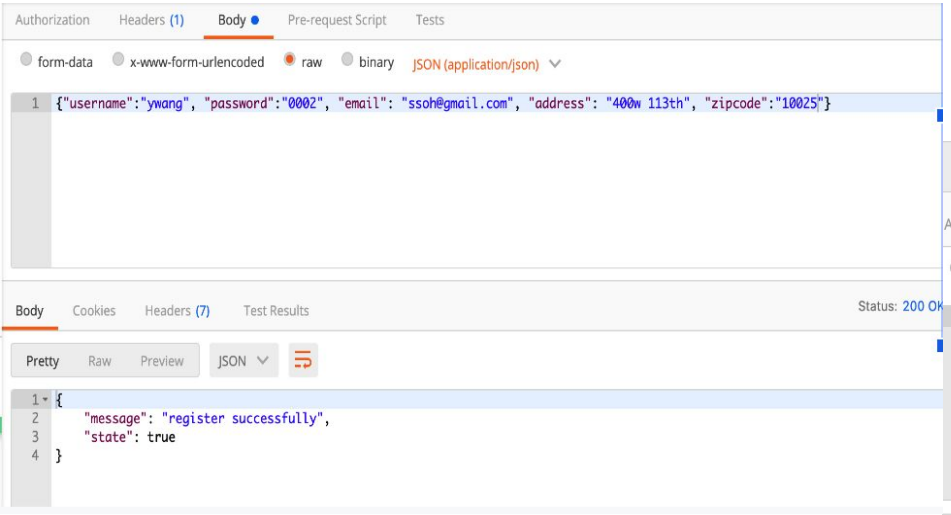


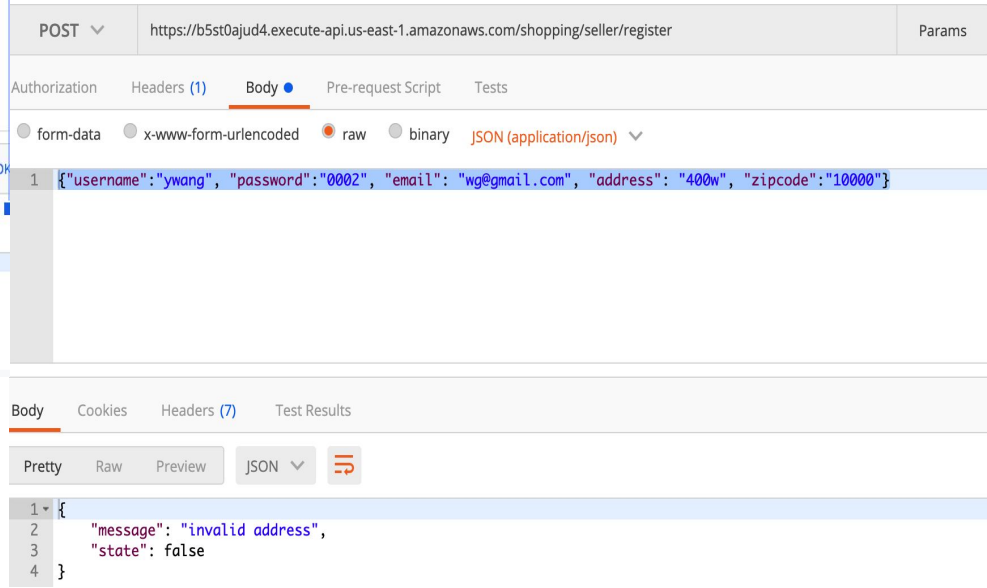


i-01838c8e3d10cbfe2 (seller)

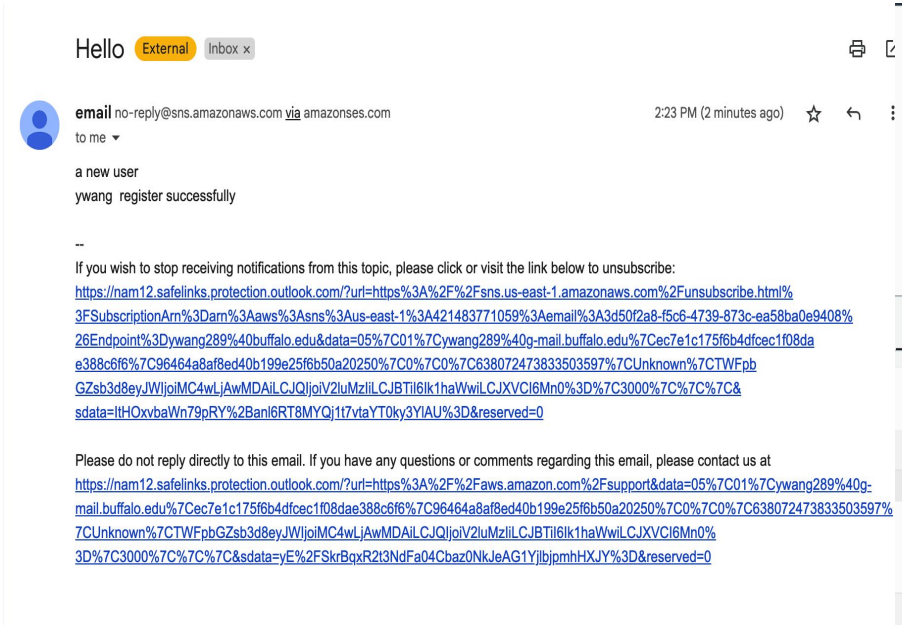
PublicIPs: 52.55.10.164 PrivateIPs: 172.31.88.37

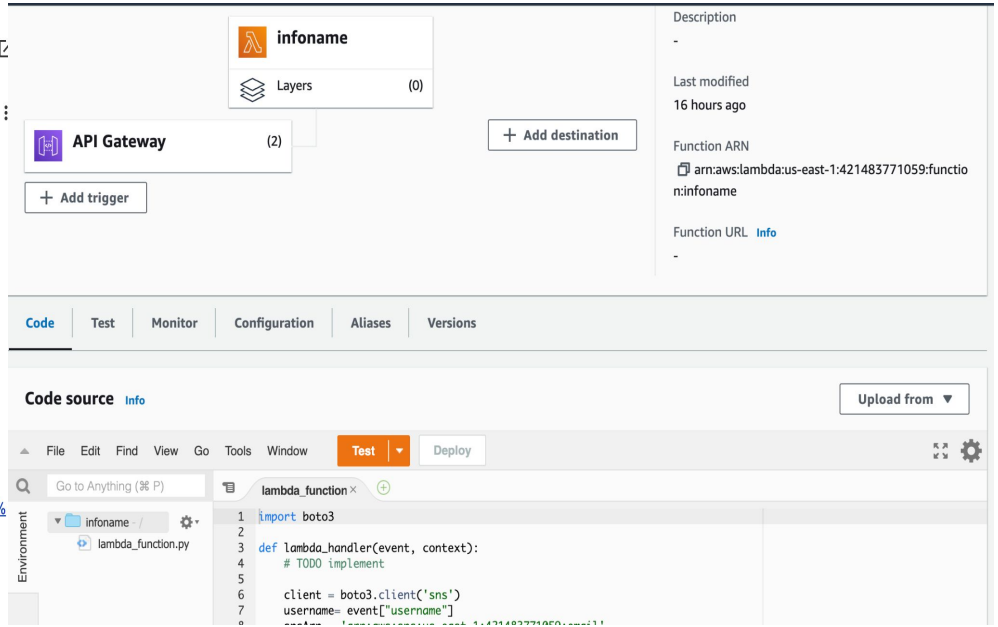
Seller service: Address Verification with SmartyStreets





Seller service:
Notification with SNS and Lambda functions





Like customer service...

Pagination

Google login

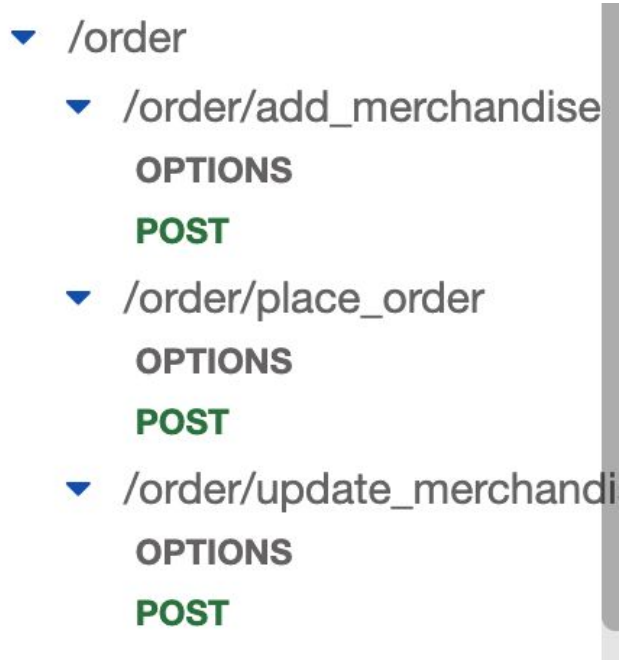
Notification with sns and lambda functions

Middlewire

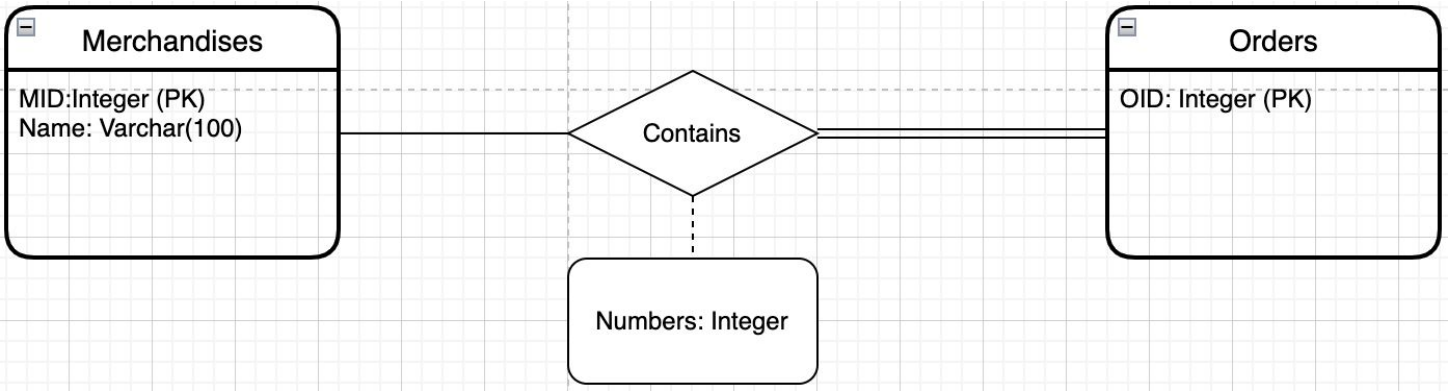
Order service

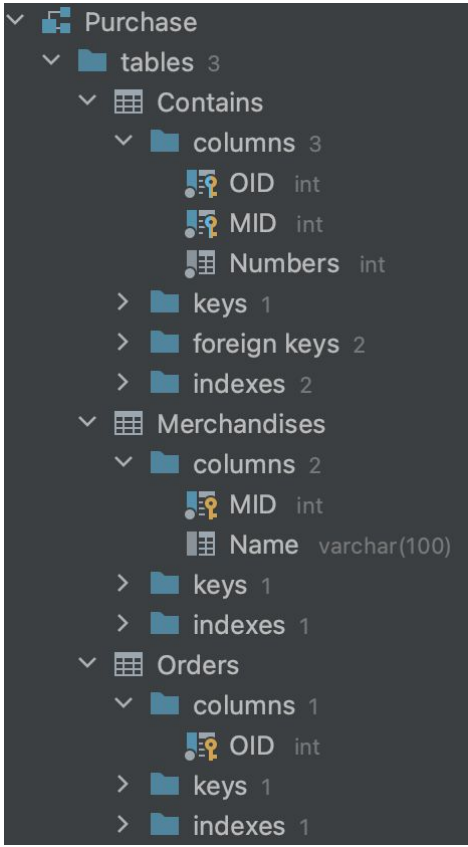
Order Service Overview

- Accept and respond with JSON
- Deploy on Elastic Beanstalk
- Connect a cloud database and browser application
- Interact with cloud database



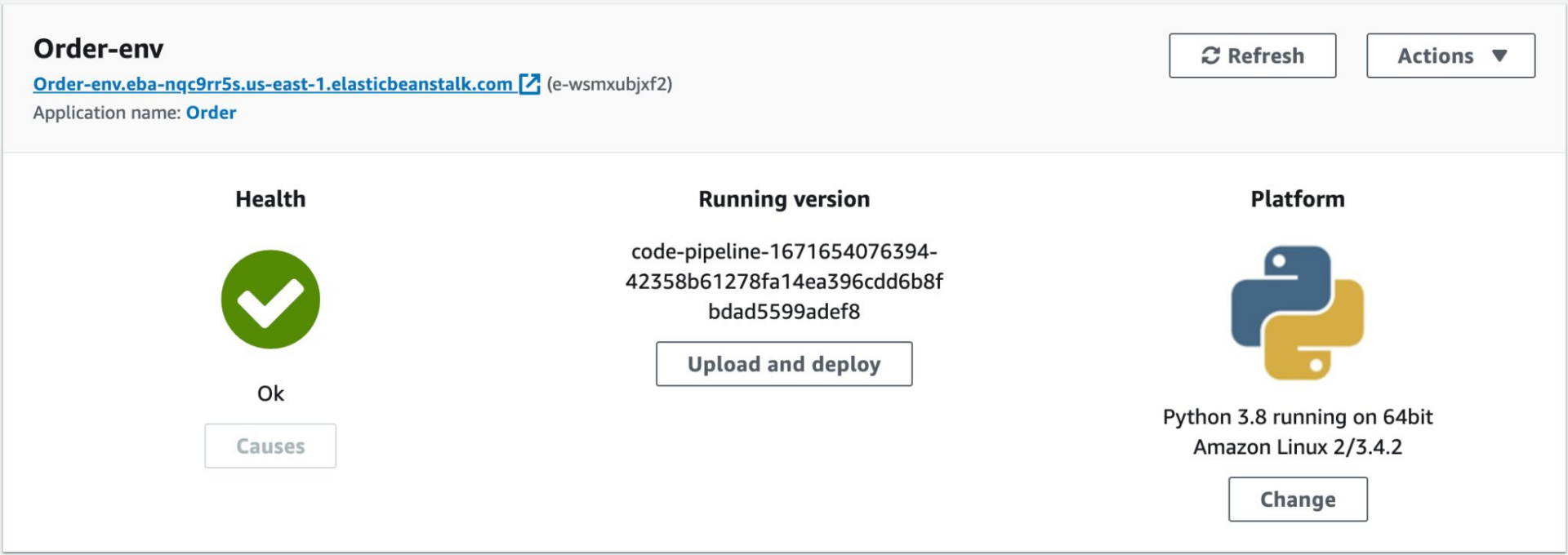
Order Service ER Diagram And Schema





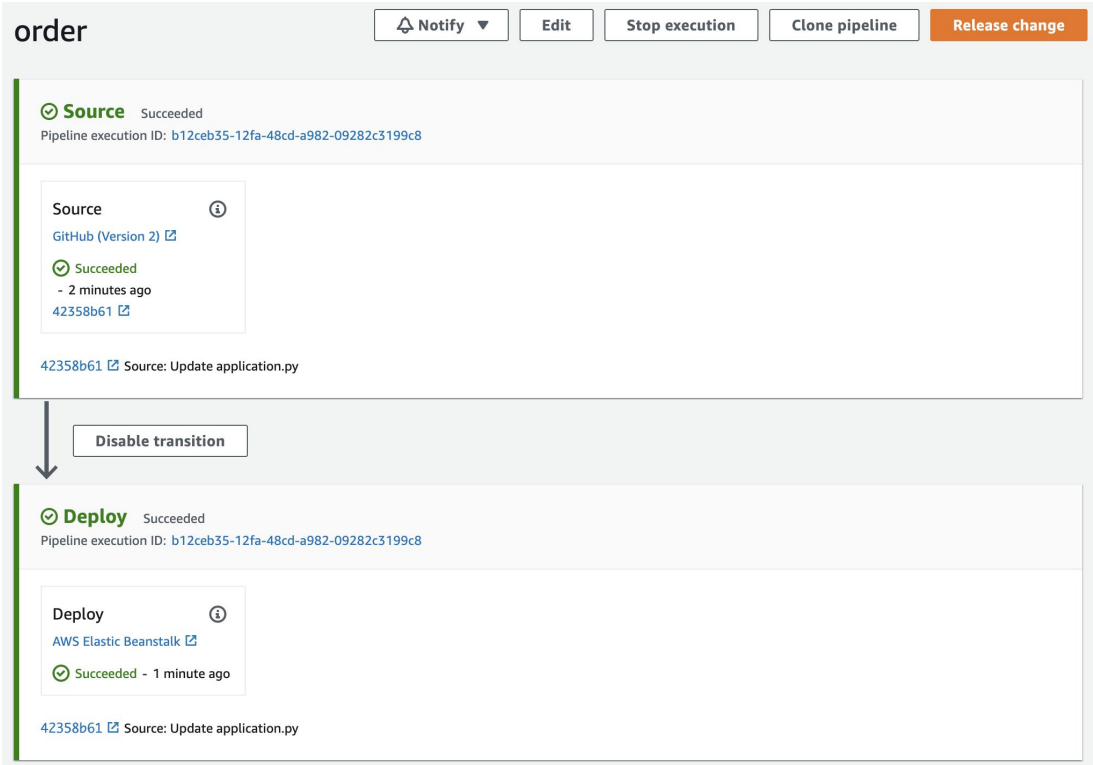
Github for order https://github.com/ywang289/order.git

Order Service: Elastic Beanstalk Deployment



Link for Elastic Beanstalk: http://order-env.eba-nqc9rr5s.us-east-1.elasticbeanstalk.com/

Elastic Beanstalk Automate Deployment /Redeployment With CodePipeline



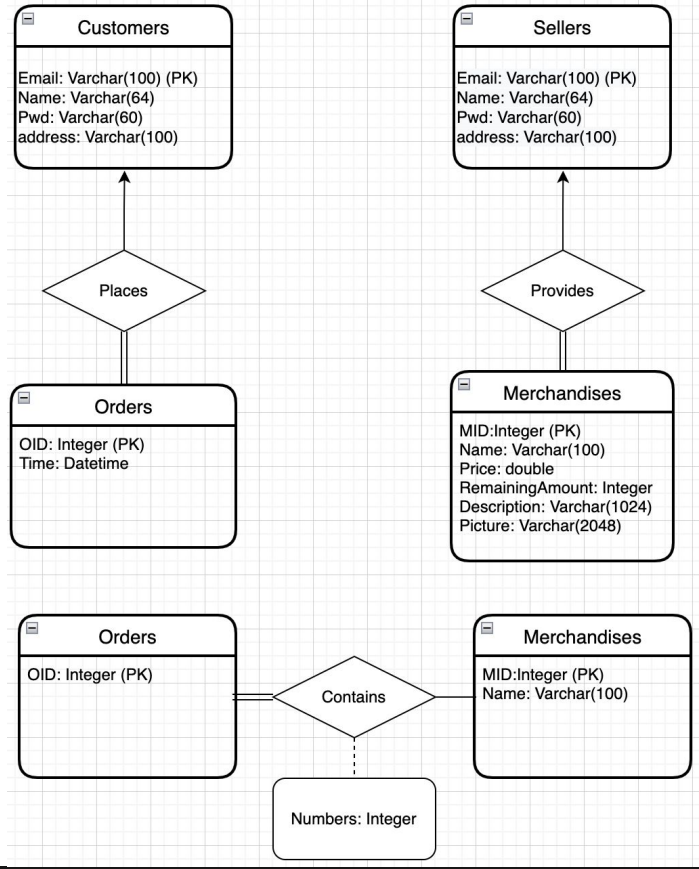
Github for CodePipeline: https://github.com/ZYbunny612/flask-example.git

(Same code as order, just add gitignore and change name to application.py for deployment)

Composite Service

Composite Service

1. Customer service
2. Seller service
3. Order service



Github for compose:
https://github.com/ywang289/compose.git

```
customer_http="http://ec2-44-201-86-144.compute-1.amazonaws.com:8080/"
seller_http="http://ec2-52-55-10-164.compute-1.amazonaws.com:8081/"
order_http="http://order-env.eba-nqc9rr5s.us-east-1.elasticbeanstalk.com/"
```

Composite Service: Sequential

1. Customer Place Order

For each involved database:
`/order/check_amount`
Sellers → check remaining amount for each purchased merchandise:
    If all remaining amounts are enough for the order, then update (decrease by the purchased numbers) for each;
    If one is not enough, then return warning message;
If check amount is successful:
    `/customer/place_order`
    Customers → Generate OID; Insert to Orders; Insert to Places; Return OID;
    If get OID:
        `/order/place_order`
        Purchases → Insert OID to Orders; for each purchased merchandise, insert OID, its MID and numbers to Contains

```python
@app.route('/customer/purchase', methods=['POST'])
def customer_purchase():
    if request.method == 'POST':
        data = json.loads(request.get_data())
        email = data['email']
        time=data['timestamp']
        items= data["items"]

        sellers_check = requests.post(seller_http+'order/check_amount', data=request.get_data())
        #{response: success or fail}
        # print(json.loads(sellers_check.text)['state'])
        # print(json.loads(sellers_check.text)['message'])
        if json.loads(sellers_check.text)['state']:
            # success
            #{email: string, timestamp: time,( current time),items:dictionary{merchandise id: amount
            print("success")
            s= json.dumps({'email':email, 'timestamp':time, 'items':items})

            get_oid = requests.post(customer_http+'customer/place_order', data=s)
            # true/ false
            oid = json.loads(get_oid.text)['oid']
            if json.loads(get_oid.text)['state']:
                print("successfully")
                # { "email":"test3@gmail.com", "timestamp":"2022-12-14 17:30:00" ,"order":{"1":"10",
                s= json.dumps({'email':email, 'timestamp':time, 'items':items, "oid": oid})
                insert_item= requests.post(order_http+'order/place_order', data=s)
    return insert_item.text
```

Composite Service: Sequential

2. Seller Add Merchandise

For each involved database:

`/seller/insert_item`

Sellers → Generate MID; Insert to Merchandises and Provides; Return MID

If get MID:

`/order/add_merchandise`

Purchases → Insert MID, name to Merchandises

```python
@app.route('/seller/insert_merchandise', methods=['POST'])
def insert_merchandise():
    if request.method == 'POST':
        data = json.loads(request.get_data())
        email = data['email']
        name= data["name"]
        price= data["price"]
        remaining_amount= data['remaining_amount']
        description= data["description"]
        picture= data['picture']
        sellers_insert= requests.post(seller_http+'seller/insert_item', data=request.get_data())

        if json.loads(sellers_insert.text)['state']:
            # success
            print(json.loads(sellers_insert.text))
            mid = json.loads(sellers_insert.text)['mid']

            #{email: string, timestamp: time,( current time),items:dictionary{merchandise id: amount}
            print("success")
            s= json.dumps({'mid':mid, "name": name})
            print ({'mid':mid, "name": name})
            insert_order = requests.post(order_http+'order/add_merchandise', data=s)

            # true/ false
            print("second")
    return insert_order.text
```

Composite Service: Sequential

3. Seller Update Merchandise

For each involved database:

`/seller/update_item`

Sellers → Update related attributes to Merchandises by MID

`/order/update_merchandise`

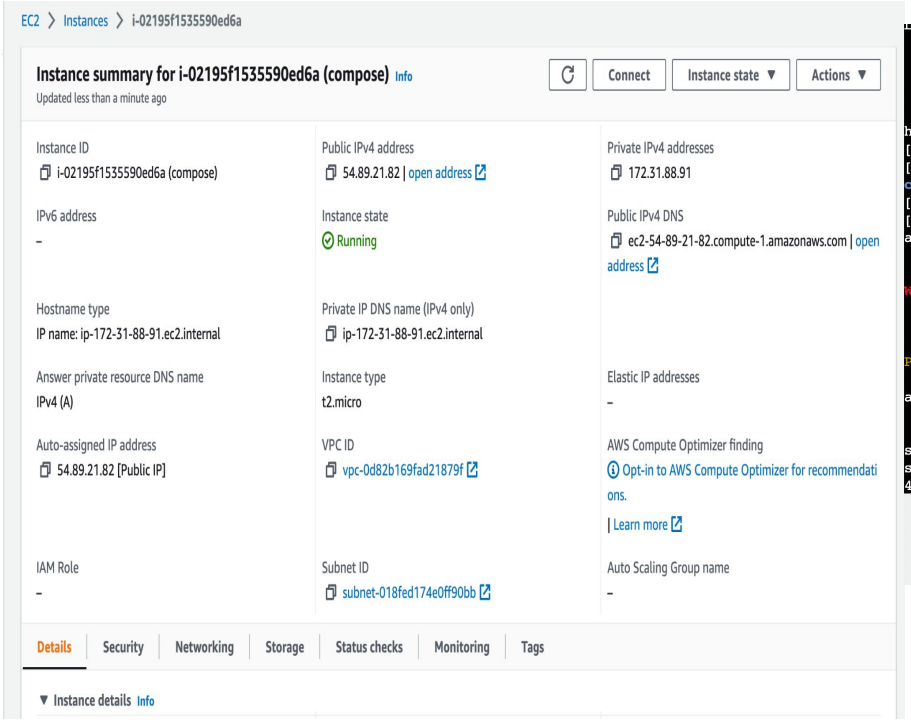
Purchases → Update name to Merchandise by MID

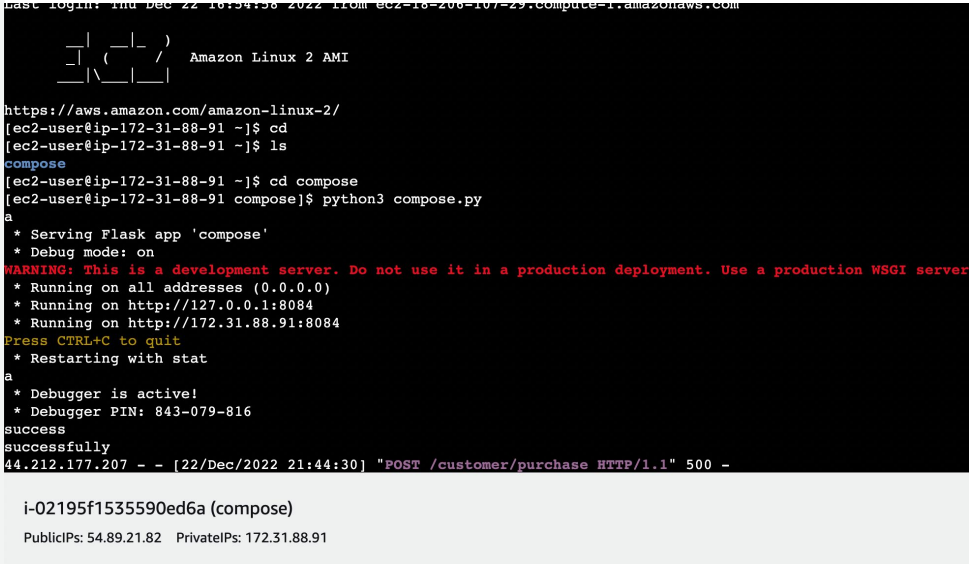
Note: seller delete merchandise is not composite

We don’t delete merchandise in the Purchase database in case that a deleted merchandise was purchased and needed to be checked details through the order

```python
#{email, name, price, remaining_amount, description, picture, mid}
@app.route('/compose/update_merchandise', methods=['POST'])
def order_detail():
    if request.method == 'POST':

        order_detail = requests.post(seller_http+'seller/update_item', data=request.get_data())
        order=json.loads(order_detail.text)
    # if can not search order, need to add something
        compose_detail= requests.post(order_http+'/order/update_merchandise', data=request.get_data())

        return json.loads(compose_detail.text)
```

EC2





Composite service:https://ec2-54-89-21-82.compute-1.amazonaws.com

Contribution

Frontend:

Juntian Sun (js5940):

1. Write the React frontend for the project
2. Create the Github Repo for the frontend
3. Deploy the React files in S3 bucket
4. Connect the frontend to API Gateway
5. Encapsulate the S3 bucket into CloudFront
6. Add Google OAuth for login on frontend

Zijian Zhang (zz2994):

1. Assist implementing the front-end project
2. Help creating queries for front-end connection
3. Use css to simply beautify the UI interface

Backend:

Yifan Wang (yw3914):

1. Write backend of microservers
2. Deploy the customer, seller, compose server in EC2
3. SNS and Lambda functions (middlewire）
4. Api gateway and cloud api for register
5. Pagination and composition service

Yi Zhang (yz4339):

1. Design, create, and manage the databases
2. Write queries needed by the backend
3. Write the backend together with Yifan Wang
4. Write the S3 and Elastic Beanstalk document
5. Deploy the order service on Elastic Beanstalk with CodePipeline

Thank you for watching!