# DISTRIBUTED SYSTEMS ASSIGNMENT REPORT

LOGO

**ASSIGNMENT REPORT**

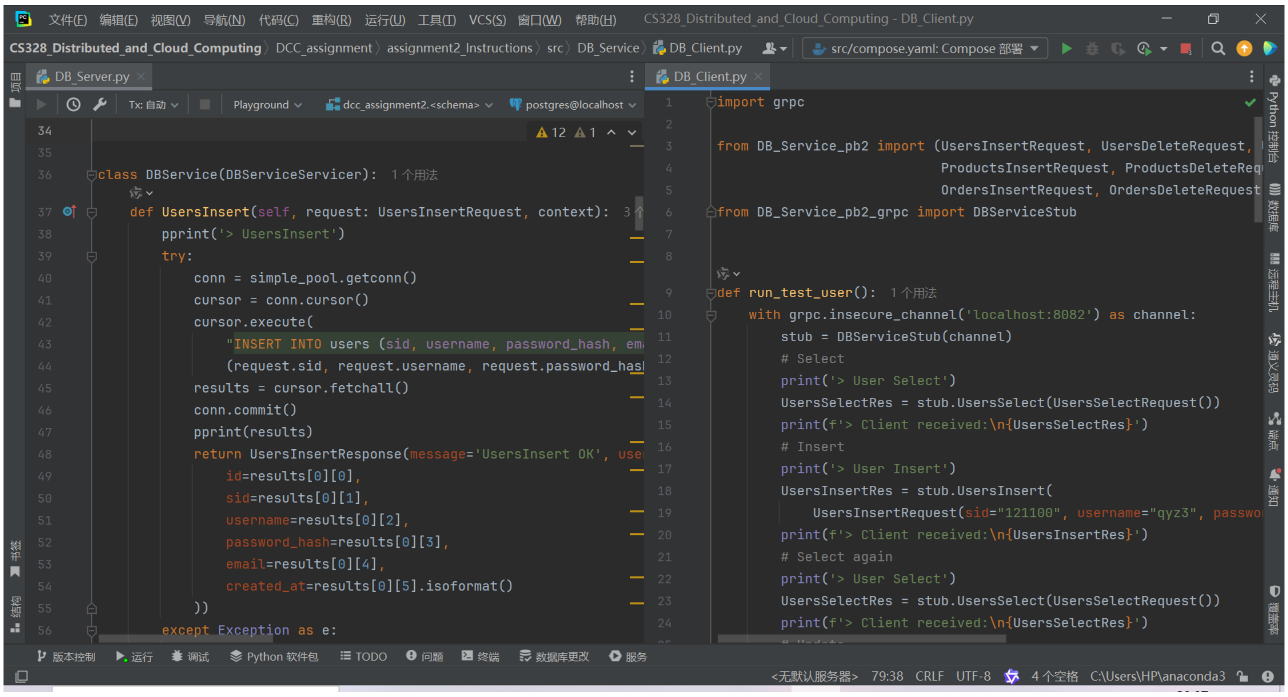
**Assignment ID: Assignment2 - SUSTech Merch Store**

**Student Name: 王谦益**

**Student ID: 12111003**

## DESIGN

- **DB Service**
  - *init.sql*
  - *DB_Service.proto*
    - three message to difine Users, Products and Orders.
    - Insert, Delete, Select, ans Update for Users, Products and Orders.
  - *DB_Service_pb2.py*, *DB_Service_pb2.pyi*, *DB_Service_pb2_grpc.py*
  - *DB_Service.py*: DBService implement all the functions.
  - *DB_Client.py*: test DBService's functions.



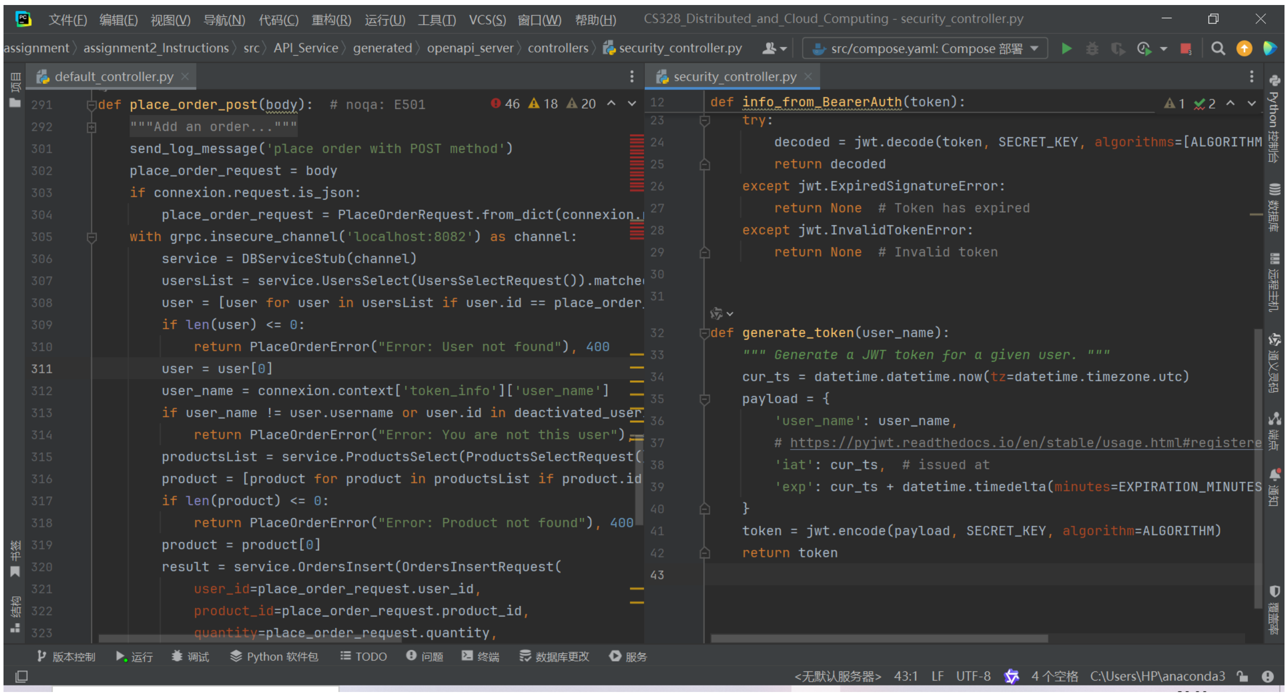
CS328_Distributed_and_Cloud_Computing - DB_Client.py
CS328_Distributed_and_Cloud_Computing 〉 DCC_assignment 〉 assignment2_Instructions 〉 src 〉 DB_Service 〉 DB_Client.py
src/compose.yaml: Compose 部署
DB_Server.py
Tx: 自动
Playground
dcc_assignment2.<schema>
postgres@localhost
DB_Client.py

```
class DBService(DBServiceServicer):  1 个用法
    def UsersInsert(self, request: UsersInsertRequest, context):
        pprint('> UsersInsert')
        try:
            conn = simple_pool.getconn()
            cursor = conn.cursor()
            cursor.execute(
                "INSERT INTO users (sid, username, password_hash, em
                (request.sid, request.username, request.password_has
            results = cursor.fetchall()
            conn.commit()
            pprint(results)
            return UsersInsertResponse(message='UsersInsert OK', use
                id=results[0][0],
                sid=results[0][1],
                username=results[0][2],
                password_hash=results[0][3],
                email=results[0][4],
                created_at=results[0][5].isoformat()
            ))
        except Exception as e:
```

```
import grpc

from DB_Service_pb2 import (UsersInsertRequest, UsersDeleteRequest,
                            ProductsInsertRequest, ProductsDeleteReq
                            OrdersInsertRequest, OrdersDeleteRequest
from DB_Service_pb2_grpc import DBServiceStub


def run_test_user():  1 个用法
    with grpc.insecure_channel('localhost:8082') as channel:
        stub = DBServiceStub(channel)
        # Select
        print('> User Select')
        UsersSelectRes = stub.UsersSelect(UsersSelectRequest())
        print(f'> Client received:\n{UsersSelectRes}')
        # Insert
        print('> User Insert')
        UsersInsertRes = stub.UsersInsert(
            UsersInsertRequest(sid="121100", username="qyz3", passwo
        print(f'> Client received:\n{UsersInsertRes}')
        # Select again
        print('> User Select')
        UsersSelectRes = stub.UsersSelect(UsersSelectRequest())
        print(f'> Client received:\n{UsersSelectRes}')
```

- **Logging Server**
  - *Logging_Service.proto*
    - two message to difine Logs.
  - *Logging_Service_pb2.py*, *Logging_Service_pb2.pyi*, *Logging_Service_pb2_grpc.py*
  - *Logging_Service.py*: LoggingService implement all the functions.
  - *Logging_Client.py*: test LoggingService's functions.

- **API Server**
  - *assignment2.yaml*: 12 paths provided.
    - A Greeting API that returns a welcome message at the base URL
    - list-products and get-product operations for products
    - register, deactivate-user, get-user, update-user, login for users
    - place-order, cancel-order, get-order for orders
  - *genetated openAPI: openapi_server*
    - security_controller.py: info_from_BearerAuth & generate_token
    - default_controller.py: 12 APIs provided to get data & 1 function to send log message.


文件(F) 编辑(E) 视图(V) 导航(N) 代码(C) 重构(R) 运行(U) 工具(T) VCS(S) 窗口(W) 帮助(H)
CS328_Distributed_and_Cloud_Computing - security_controller.py
assignment 〉 assignment2_Instructions 〉 src 〉 API_Service 〉 generated 〉 openapi_server 〉 controllers 〉 security_controller.py
src/compose.yaml: Compose 部署
default_controller.py
security_controller.py
版本控制 运行 调试 Python 软件包 TODO 问题 终端 数据库更改 服务
<无默认服务器> 43:1 LF UTF-8 4 个空格 C:\Users\HP\anaconda3

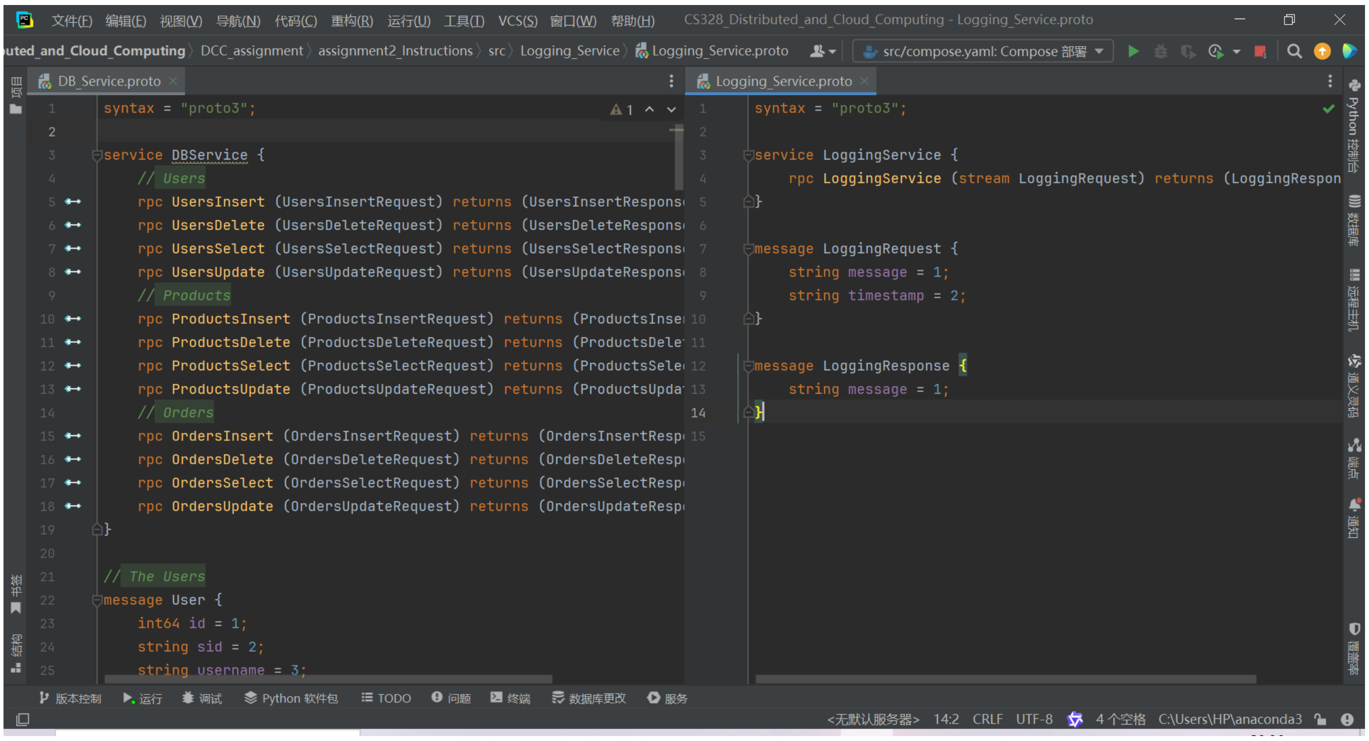
```python
def place_order_post(body):  # noqa: E501
    """Add an order..."""
    send_log_message('place order with POST method')
    place_order_request = body
    if connexion.request.is_json:
        place_order_request = PlaceOrderRequest.from_dict(connexion.
    with grpc.insecure_channel('localhost:8082') as channel:
        service = DBServiceStub(channel)
        usersList = service.UsersSelect(UsersSelectRequest()).matche
        user = [user for user in usersList if user.id == place_order
        if len(user) <= 0:
            return PlaceOrderError("Error: User not found"), 400
        user = user[0]
        user_name = connexion.context['token_info']['user_name']
        if user_name != user.username or user.id in deactivated_user
            return PlaceOrderError("Error: You are not this user")
        productsList = service.ProductsSelect(ProductsSelectRequest(
        product = [product for product in productsList if product.id
        if len(product) <= 0:
            return PlaceOrderError("Error: Product not found"), 400
        product = product[0]
        result = service.OrdersInsert(OrdersInsertRequest(
            user_id=place_order_request.user_id,
            product_id=place_order_request.product_id,
            quantity=place_order_request.quantity,
```

```python
def info_from_BearerAuth(token):
    try:
        decoded = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM
        return decoded
    except jwt.ExpiredSignatureError:
        return None  # Token has expired
    except jwt.InvalidTokenError:
        return None  # Invalid token


def generate_token(user_name):
    """ Generate a JWT token for a given user. """
    cur_ts = datetime.datetime.now(tz=datetime.timezone.utc)
    payload = {
        'user_name': user_name,
        # https://pyjwt.readthedocs.io/en/stable/usage.html#registere
        'iat': cur_ts,  # issued at
        'exp': cur_ts + datetime.timedelta(minutes=EXPIRATION_MINUTES
    }
    token = jwt.encode(payload, SECRET_KEY, algorithm=ALGORITHM)
    return token
```

# QUESTIONS IN PDF

## 1. IMPLEMENTATION

- **DB Service**
  1. write `DB_Service.proto` and set response type & request type for each CRUD operations for users, products and orders.
  2. run command to generate `DB_Service_pb2.py`, `DB_Service_pb2.pyi`, `DB_Service_pb2_grpc.py`.
  3. write `DB_Service.py` to implement all the CRUD functions with exception handling.
  4. write `DB_Client.py` to test DBService's functions.


CS328_Distributed_and_Cloud_Computing - Logging_Service.proto
DB_Service.proto
Logging_Service.proto
src/compose.yaml: Compose 部署

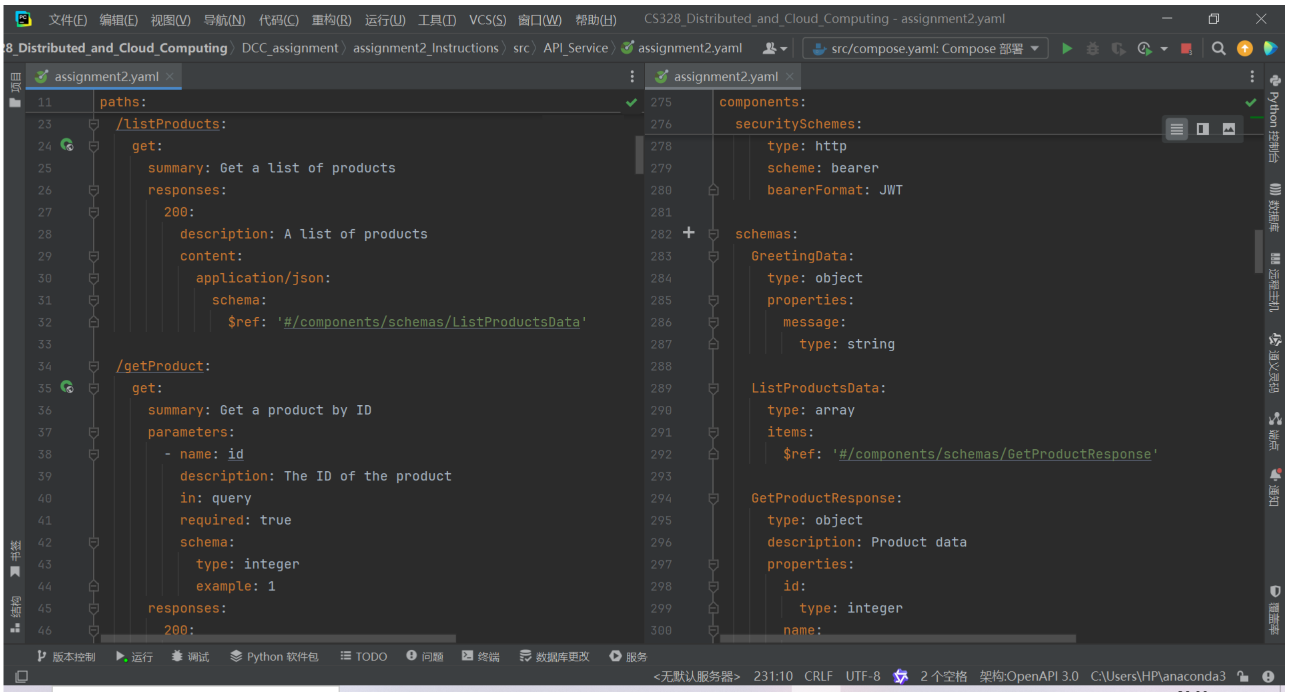
```
syntax = "proto3";

service DBService {
    // Users
    rpc UsersInsert (UsersInsertRequest) returns (UsersInsertRespons
    rpc UsersDelete (UsersDeleteRequest) returns (UsersDeleteRespons
    rpc UsersSelect (UsersSelectRequest) returns (UsersSelectRespons
    rpc UsersUpdate (UsersUpdateRequest) returns (UsersUpdateRespons
    // Products
    rpc ProductsInsert (ProductsInsertRequest) returns (ProductsInse
    rpc ProductsDelete (ProductsDeleteRequest) returns (ProductsDele
    rpc ProductsSelect (ProductsSelectRequest) returns (ProductsSele
    rpc ProductsUpdate (ProductsUpdateRequest) returns (ProductsUpda
    // Orders
    rpc OrdersInsert (OrdersInsertRequest) returns (OrdersInsertResp
    rpc OrdersDelete (OrdersDeleteRequest) returns (OrdersDeleteResp
    rpc OrdersSelect (OrdersSelectRequest) returns (OrdersSelectResp
    rpc OrdersUpdate (OrdersUpdateRequest) returns (OrdersUpdateResp
}

// The Users
message User {
    int64 id = 1;
    string sid = 2;
    string username = 3;
```

```
syntax = "proto3";

service LoggingService {
    rpc LoggingService (stream LoggingRequest) returns (LoggingRespon
}

message LoggingRequest {
    string message = 1;
    string timestamp = 2;
}

message LoggingResponse {
    string message = 1;
}
```

- **Logging Server**
    1. write `Logging_Service.proto` and set response type & request type for logs.
    2. run command to generate `Logging_Service_pb2.py`, `Logging_Service_pb2.pyi`, `Logging_Service_pb2_grpc.py`.
    3. write `Logging_Service.py` to implement aynsc function to send log message to the logging server.
    4. write `Logging_Client.py` to test LoggingService's functions.
- **API Server**
    1. write `assignment2.yaml` to define 12 paths provided, the request and response type for each path.
    2. run command to generate `openapi_server`.
    3. write `security_controller.py` to implement the token part.
    4. write `default_controller.py` to implement 12 APIs provided to get data & 1 function to send log message.

## 2. AUTHENTICATION

- **APIs require authentication**
    - deactivate-user: /deactivateUser
    - update-user: /updateUser
    - place-order: /placeOrder
    - cancel-order: /cancelOrder
    - get-order: /getOrder


CS328_Distributed_and_Cloud_Computing - assignment2.yaml
assignment2.yaml
src/compose.yaml: Compose 部署
<无默认服务器> 231:10 CRLF UTF-8 2 个空格 架构:OpenAPI 3.0 C:\Users\HP\anaconda3

```yaml
paths:
  /listProducts:
    get:
      summary: Get a list of products
      responses:
        200:
          description: A list of products
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ListProductsData'

  /getProduct:
    get:
      summary: Get a product by ID
      parameters:
        - name: id
          description: The ID of the product
          in: query
          required: true
          schema:
            type: integer
            example: 1
      responses:
        200:
```

```yaml
components:
  securitySchemes:
      type: http
      scheme: bearer
      bearerFormat: JWT

  schemas:
    GreetingData:
      type: object
      properties:
        message:
          type: string

    ListProductsData:
      type: array
      items:
        $ref: '#/components/schemas/GetProductResponse'

    GetProductResponse:
      type: object
      description: Product data
      properties:
        id:
          type: integer
        name:
```

- **implement**

    1. set `security` for each path in `assignment2.yaml`.
    2. write `info_from_BearerAuth` and `generate_token` in `security_controller.py`.
    3. use `user_name = connexion.context['token_info']['user_name']` to get the user name from the token and compare it with the user info given by the request.

## 3. DATA TYPES

- **integer**: numbers like id, quantity
- **float**: decimal numbers like price, total_price
- **string**: varchar like username, email
- **array**: list when select data with multiple results
- **object**: dict when select data with one result and multiple fields

## 4. PROTO MESSAGE ENCODING

- proto message:

```
message LoggingRequest {
  string message = 1;
  string timestamp = 2;
}
```

```
message   = "get product with GET method" - 29 bytes
timestamp = 1733057906.1217306           - 19 bytes
```

- encoding:

- field 1

```
[0x0A] (Tag 1 with wire type 2)
[0x1D] (Length 29)
[0x67 0x65 0x74 0x20 0x70 0x72 0x6F 0x64 0x75 0x63 0x74 0x20 0x77 0x69
0x74 0x68 0x20 0x47 0x45 0x54 0x20 0x6D 0x65 0x74 0x68 0x6F 0x64]
```

- field 2

```
[0x12] (Tag 2 with wire type 2)
[0x13] (Length 19)
[0x31 0x37 0x33 0x33 0x30 0x35 0x37 0x39 0x30 0x36 0x2E 0x31 0x32 0x31
0x37 0x33 0x30 0x36]
```

## 5. STREAMING PRC

1. write `Logging_Service.proto`, set response message with {meaage, timestamp} and request message with {message}.
2. run command to generate `Logging_Service_pb2.py`, `Logging_Service_pb2.pyi`, `Logging_Service_pb2_grpc.py`.
3. write `Logging_Service.py` with `async def LoggingService(self, request_iterator: Iterator[LoggingRequest], context)` to receive the message iteratorly and `produce(topic, msg)` to kafka.

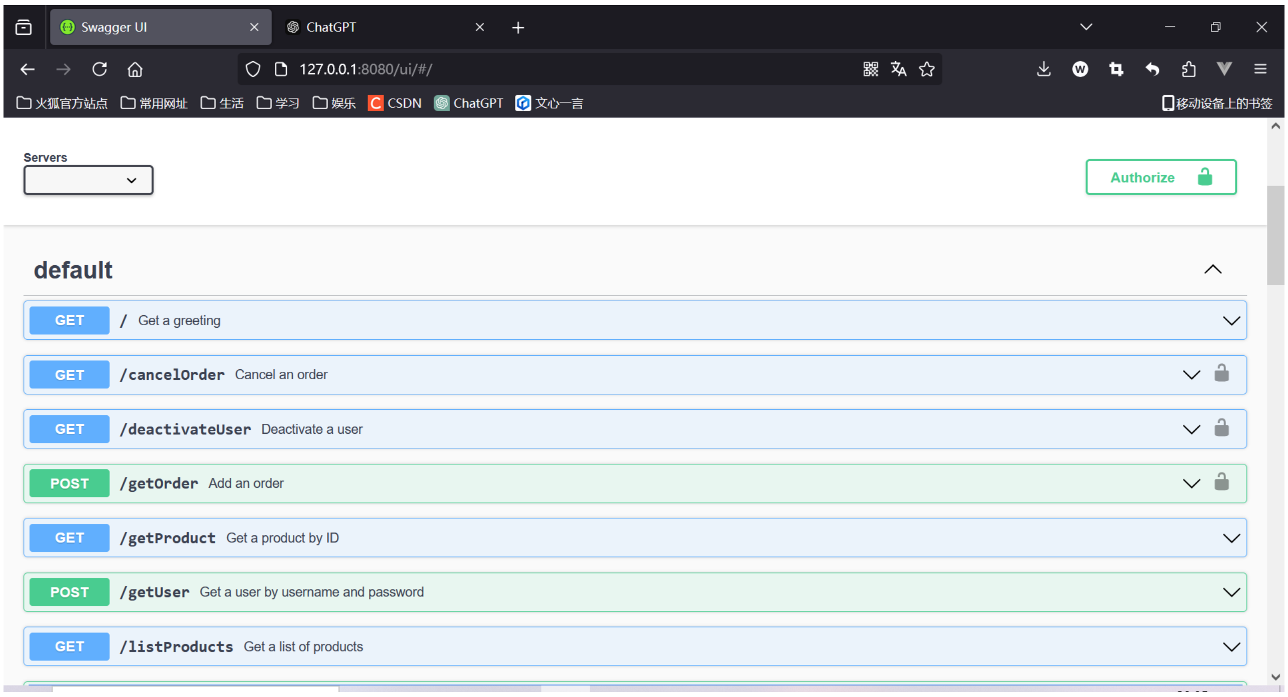
## 6. DOCKER

1. define services in a docker-compose.yml file `compose.yaml`.
2. figeure out key elements of docker compose configuration.
   - services
   - volumes
   - networks
   - ports
   - depends_on
   - ...
3. run command to build and run the docker compose.

## 7. RUN

1. `python DB_Server.py`
2. `python Logging_Server.py`
3. `python -m openapi_server`

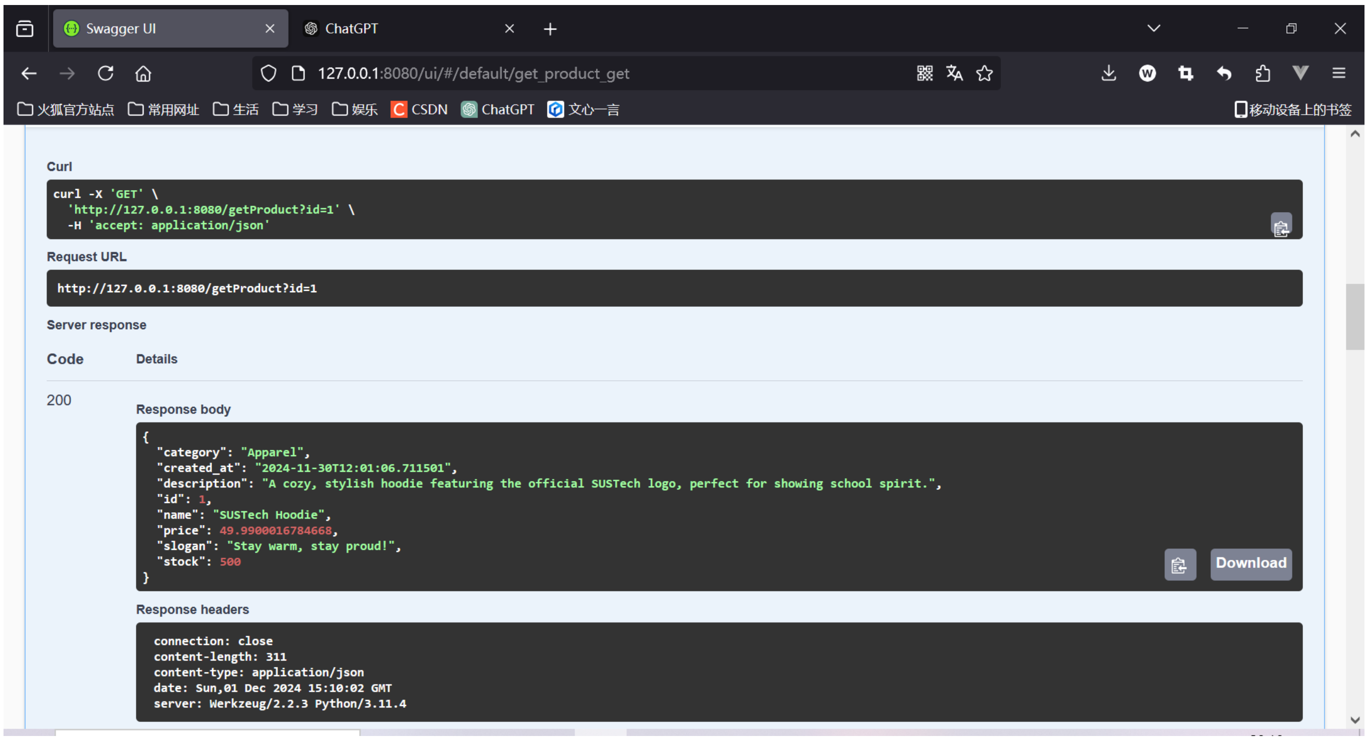
- use Swagger UI and `logging.info()` fuction to test APIs.



GET / Get a greeting
GET /cancelOrder Cancel an order
GET /deactivateUser Deactivate a user
POST /getOrder Add an order
GET /getProduct Get a product by ID
POST /getUser Get a user by username and password
GET /listProducts Get a list of products

- write `print()` function to show result for each step or fuction.

## RUNNING RESULTS



Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8080/getProduct?id=1' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8080/getProduct?id=1
```

Server response

Code Details

200 Response body

```
{
  "category": "Apparel",
  "created_at": "2024-11-30T12:01:06.711501",
  "description": "A cozy, stylish hoodie featuring the official SUSTech logo, perfect for showing school spirit.",
  "id": 1,
  "name": "SUSTech Hoodie",
  "price": 49.9900016784668,
  "slogan": "Stay warm, stay proud!",
  "stock": 500
}
```

Response headers

```
 connection: close
 content-length: 311
 content-type: application/json
 date: Sun,01 Dec 2024 15:10:02 GMT
 server: Werkzeug/2.2.3 Python/3.11.4
```

```
INFO:root:Server started on port 50051
INFO:root:Received logging request.
INFO:root:Produced message: Received message: get product with GET method sent at time = 1733065801.6440403
```

```
 'Drinkware',
 Decimal('19.99'),
 'Hydrate with pride!',
 500,
 datetime.datetime(2024, 11, 30, 12, 1, 6, 711501)),
(3,
 'SUSTech Notebook',
 'A premium notebook with the SUSTech logo, ideal for jotting down ideas, '
 'notes, and memories.',
 'Stationery',
 Decimal('9.99'),
 'Write your future with SUSTech.',
 500,
 datetime.datetime(2024, 11, 30, 12, 1, 6, 711501)),
(9,
 '111',
 'hhh',
 'sb',
 Decimal('12.00'),
 'cnm',
 10,
 datetime.datetime(2024, 12, 1, 6, 30, 18, 132893))]
```

## PROBLEMS

when I implement the `Logging_Service.py`, I got the error:

```
  INFO:root:Server started on port 50051
  ERROR:grpc._cython.cygrpc:Unexpected [TypeError] raised by servicer method
[/Service/LoggingService]
  Traceback (most recent call last):
    File "src\\python\\grpcio\\grpc\\_cython\\_cygrpc/aio/server.pyx.pxi", line
689, in grpc._cython.cygrpc._handle_exceptions
    File "src\\python\\grpcio\\grpc\\_cython\\_cygrpc/aio/server.pyx.pxi", line
845, in _handle_rpc
    File "src\\python\\grpcio\\grpc\\_cython\\_cygrpc/aio/server.pyx.pxi", line
645, in _handle_stream_unary_rpc
    File "src\\python\\grpcio\\grpc\\_cython\\_cygrpc/aio/server.pyx.pxi", line
408, in _finish_handler_with_unary_response
  TypeError: object async_generator can't be used in 'await' expression
```

I solved it by remove `yield LoggingResponse(message='Hello from Logging Service!')`

also I got the error:

```
  TypeError: descriptor 'SerializeToString' for 'google._upb._message.Message'
objects doesn't apply to a 'NoneType' object
```

this error is caused by the `None` value returned by service for no `return` statement, so I added `return None` in the end of the function `return LoggingResponse(message='Logging service stopped.')` in `finally` block.