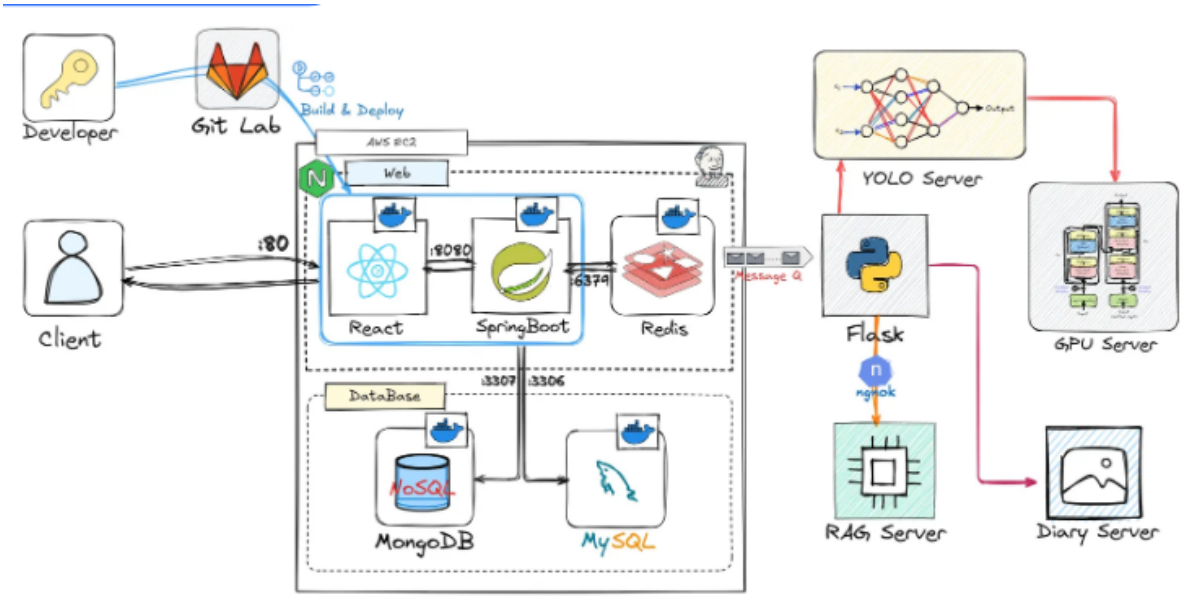
포팅 메뉴얼

작성일시	@2024년 8월 16일 오전 11:04
복습	☐

1. 서비스 파이프 라인



설치 환경

```
os : ubuntu 20.04
IDE : Intellij
            vscode
node : 20.1
jdk : 17
mongoDB : 7.0.12
 - Port : 27017/tcp
mySQL : 3.8
    - Port : 3306/tcp
Docker : 20
Python : 3.10.12
jenkins : lts
flask : latest
```

```
    - Port : 33333/tcp, 37777/tcp
redis : latest
    - Port : 6379/tcp
react : nginx 기반
    - Port: 80/tcp, 443/tcp
Spring : 8805/tcp
```

2. Front CI/CD

- jenkins
  - CI

```
echo docker login
docker login -u ${DOCKER_USER} -p ${DOCKER_PASSWORD} docke

echo docker image build
docker build -t doki2580/feedme-front:latest \
  --build-arg REACT_APP_API_KEY=8743f3d92bcccf8dccf0c31f38
  -f ./Front-feedme/Dockerfile ./Front-feedme

echo docker image push
docker push doki2580/feedme-front:latest

# 기존 도커 이미지 삭제를 위한 준비
echo Checking for dangling images
no_tag_image_ids=$(docker images -f "dangling=true" -q)

if [ -z "$no_tag_image_ids" ]; then
  echo "No dangling images to remove."
else
  echo "Removing dangling images"

  # 각 이미지 ID에 대해 순회하며 삭제하기 전에 해당 이미지의 컨테이너를
  for image_id in $no_tag_image_ids; do
    # 해당 이미지 ID를 사용하는 컨테이너 중지 및 삭제
    container_ids=$(docker ps -a -q --filter "ancestor=$im
```

```
    if [ -n "$container_ids" ]; then
      for container_id in $container_ids; do
        docker stop $container_id
        docker rm $container_id
      done
    fi

    # 이미지 삭제
    docker rmi $image_id
    echo "Deleted image $image_id"
  done
fi
```

- CD

```
# 이미지 이름과 태그
IMAGE_NAME="doki2580/feedme-front"
TAG="latest"

# 새로운 이미지 pull
docker pull $IMAGE_NAME:$TAG

# 기존 컨테이너 stop 및 remove
docker stop feedme-front
docker rm feedme-front

# 새로운 이미지로 컨테이너 시작
docker run -d -p 80:80 -p 443:443 \
-v /home/ubuntu/certbot/etc/letsencrypt:/etc/letsencrypt \
-v /home/ubuntu/certbot/www:/var/www/certbot \
-v /home/ubuntu/nginx/conf/default.conf:/etc/nginx/conf.d/
-v /home/ubuntu/env/front/.env:/.env \
--env-file /home/ubuntu/env/front/.env \
--network web-network \
--name feedme-front $IMAGE_NAME:$TAG

# docker network connect web-network feedme-front
```

```
# 기존 도커 이미지 삭제
no_tag_image_ids=$(docker images -f "dangling=true" -q)

if [ -z "$no_tag_image_ids" ]; then
  echo "삭제할 이미지가 없습니다."
else
  # 각 이미지 ID에 대해 순회하며 삭제하기
  for image_id in $no_tag_image_ids; do
    docker rmi $image_id
    echo "이미지 $image_id 삭제 완료"
  done
fi
```

- Docker file

```
# Docker file

# Stage 1: Build the application using Node.js
FROM node:20 AS build

# Set the working directory for the build stage
WORKDIR /app

# Define build arguments
ARG REACT_APP_API_KEY

# Set environment variables
ENV REACT_APP_API_KEY=$REACT_APP_API_KEY

# Copy the package.json and package-lock.json (if availabl
COPY package.json package-lock.json ./
RUN npm install --silent

# Copy the rest of your app's source code from your host t
COPY . .

# Build the application
RUN npm run build
```

```
# Stage 2: Serve the application using Nginx
FROM nginx:stable-alpine as production-stage

# Create a directory for Jenkins logs.
RUN mkdir -p /var/log/nginx/jenkins

# Set permissions if necessary (This step may not be neces
RUN chmod -R 755 /var/log/nginx

# Remove the default Nginx configuration
COPY ./default.conf /etc/nginx/conf.d

# Copy the built application from the build stage to the N
COPY --from=build /app/build /usr/share/nginx/html

# Ensure logging statements go to the standard output (for
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
    && ln -sf /dev/stderr /var/log/nginx/error.log

# Expose port 80 and 443 (HTTPS)
EXPOSE 80

# Start Nginx in the foreground
CMD ["nginx", "-g", "daemon off;"]
```

- default.conf

```
# WebSocket 지원을 위한 설정
map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
}

# HTTP 요청을 HTTPS로 리디렉션하는 서버 블록
server {
    listen 80;
    server_name i11b104.p.ssafy.io;
```

```
    location / {
        return 301 https://$host$request_uri;
    }
}

# HTTPS 설정을 위한 서버 블록.
server {
    listen 443 ssl;
    server_name i11b104.p.ssafy.io;

    # 로그 설정
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    # SSL 인증서 설정
    ssl_certificate /etc/letsencrypt/live/i11b104.p.ssafy.
    ssl_certificate_key /etc/letsencrypt/live/i11b104.p.ss
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;

    # 잘못된 헤더 무시 설정
    ignore_invalid_headers off;

    # Jenkins 정적 파일 처리
    location ~ "^/static/[0-9a-fA-F]{8}/(.*)$" {
        # 정적 파일 요청을 루트로 리다이렉트
        rewrite "^/static/[0-9a-fA-F]{8}/(.*)" /$1 last;
    }

    # Jenkins 사용자 콘텐츠 처리
    location /userContent {
        root /var/lib/jenkins/;  # Jenkins 사용자 콘텐츠 디렉
        if (!-f $request_filename){
            rewrite (.*) /$1 last;
            break;
        }
        sendfile on;
```

```
    }

    # 기본 웹 페이지 제공을 위한 location 블록
    location / {
        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' '*';
            add_header 'Access-Control-Allow-Methods' 'GET
            add_header 'Access-Control-Allow-Headers' 'Ori
            add_header 'Access-Control-Max-Age' 1728000;
            add_header 'Content-Type' 'text/plain charset=
            add_header 'Content-Length' 0;
            return 204;
        }
        root /usr/share/nginx/html;
        index index.html index.htm;
        charset utf-8;
        try_files $uri $uri/ /index.html;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forw
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Port $server_port;
        proxy_set_header X-Nginx-Proxy true;
    }

    # API 요청을 백엔드 서버로 프록시하는 location 블록
    location /api/ {
        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' '*';
            add_header 'Access-Control-Allow-Methods' 'GET
            add_header 'Access-Control-Allow-Headers' 'Ori
            add_header 'Access-Control-Max-Age' 1728000;
            add_header 'Content-Type' 'text/plain charset=
```

```
            add_header 'Content-Length' 0;
            return 204;
        }
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forw
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Port $server_port;

        proxy_pass http://back-server:8085/;  # Spring 백엔
    }

    # Jenkins에 대한 요청을 프록시하는 location 블록
    location /jenkins/ {
        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' '*';
            add_header 'Access-Control-Allow-Methods' 'GET
            add_header 'Access-Control-Allow-Headers' 'Ori
            add_header 'Access-Control-Max-Age' 1728000;
            add_header 'Content-Type' 'text/plain charset=
            add_header 'Content-Length' 0;
            return 204;
        }
        sendfile off;
        proxy_pass http://jenkins:8080/jenkins/;  # Jenkin
        proxy_redirect http://jenkins:8080/jenkins/ https:
        proxy_http_version 1.1;

        # Jenkins 웹소켓 에이전트 지원
        proxy_set_header Connection $connection_upgrade;
        proxy_set_header Upgrade $http_upgrade;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
```

```
        proxy_set_header X-Forwarded-For $proxy_add_x_forw
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Port $server_port;
        proxy_max_temp_file_size 0;

        # 업로드 크기 제한
        client_max_body_size 10m;
        client_body_buffer_size 128k;

        # 프록시 타임아웃 설정
        proxy_connect_timeout 90;
        proxy_send_timeout 90;
        proxy_read_timeout 90;
        proxy_request_buffering off;  # HTTP CLI 명령어 지원
    }
}
```

3. Backend CI/CD

- Jenkins
  - CI

```
echo Docker image build
docker build -t doki2580/feedme-back:latest -f ./Backend-f

echo Docker image push
docker push doki2580/feedme-back:latest

# 기존 도커 이미지 삭제를 위한 준비
echo Checking for dangling images
no_tag_image_ids=$(docker images -f "dangling=true" -q)

if [ -z "$no_tag_image_ids" ]; then
  echo "No dangling images to remove."
else
  echo "Removing dangling images"
```

```
  # 각 이미지 ID에 대해 순회하며 삭제하기 전에 해당 이미지의 컨테이너를
  for image_id in $no_tag_image_ids; do
    # 해당 이미지 ID를 사용하는 컨테이너 중지 및 삭제
    container_ids=$(docker ps -a -q --filter "ancestor=$im
    if [ -n "$container_ids" ]; then
      for container_id in $container_ids; do
        docker stop $container_id
        docker rm $container_id
      done
    fi

    # 이미지 삭제
    docker rmi $image_id
    echo "Deleted image $image_id"
  done
fi
```

- CD

```
# 이미지 이름과 태그
IMAGE_NAME="doki2580/feedme-back"
TAG="latest"

# 새로운 이미지 pull
echo "Pulling the latest image: $IMAGE_NAME:$TAG"
sudo docker pull $IMAGE_NAME:$TAG

# 기존 컨테이너 stop 및 remove
echo "Stopping and removing existing container: feedme-bac
sudo docker stop feedme-back || echo "No running container
sudo docker rm feedme-back || echo "No container to remove

# 새로운 이미지로 컨테이너 시작
echo "Starting new container: back-server with image $IMAG
sudo docker run --name back-server \
  -e SPRING_DATASOURCE_URL=jdbc:mysql://i11b104.p.ssafy.io
  -e SPRING_DATASOURCE_USERNAME=root \
  -e SPRING_DATASOURCE_PASSWORD=ssafy \
```

```
  -e SPRING_DATASOURCE_DRIVER=com.mysql.cj.jdbc.Driver \
  --env-file /home/ubuntu/env/.env \
  -v /home/ubuntu/env/.env:/app/.env \
  -d doki2580/feedme-back:latest

docker network connect web-network back-server

# 컨테이너 상태 확인
sudo docker ps -a | grep back-server

# 기존 도커 이미지 삭제
echo "Checking for dangling images to remove"
no_tag_image_ids=$(sudo docker images -f "dangling=true" -

if [ -z "$no_tag_image_ids" ]; then
  echo "No dangling images to remove."
else
  echo "Removing dangling images"
  # 각 이미지 ID에 대해 순회하며 삭제하기
  for image_id in $no_tag_image_ids; do
    sudo docker rmi $image_id
    echo "Deleted image $image_id"
  done
fi
```

- Docker file

```
server {
    listen 80;
    server_name i11b104.p.ssafy.io;
    location / {
        return 301 https://$host$request_uri;
    }
}

#server {
#    listen 8085;
#    server_name i11b104.p.ssafy.io;
```

```
#    location / {
#        return 301 https://$host$request_uri;
#    }
#}

server {
    listen 443 ssl;
    server_name i11b104.p.ssafy.io;
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    ssl_certificate /etc/letsencrypt/live/i11b104.p.ssafy.
    ssl_certificate_key /etc/letsencrypt/live/i11b104.p.ss
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        charset utf-8;
        try_files $uri $uri/ /index.html;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forw
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Nginx-Proxy true;
    }

    location /api {
        rewrite ^/api/(.*|$) /$1  break;
        proxy_pass http://back-server:8085;
        proxy_redirect off;
        charset utf-8;
```

```
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forw
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Nginx-Proxy true;
    }

    location /jenkins/ {
        proxy_pass http://jenkins:8080/;
        proxy_redirect off;
        charset utf-8;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forw
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Nginx-Proxy true;
    }


    # location /api/ {
    #     proxy_http_version 1.1;
    #     proxy_set_header Upgrade $http_upgrade;
    #     proxy_set_header Connection "upgrade";
    #     proxy_set_header Host $host;
    #     proxy_set_header X-Real-IP $remote_addr;
    #     proxy_set_header X-Forwarded-For $proxy_add_x_fo
    #     proxy_set_header X-Forwarded-Proto $scheme;

    #     proxy_pass http://back-server:8085; # back-serve
    # }
```

```
    # location /jenkins/ {
    #     proxy_set_header Host $host;
    #     proxy_set_header X-Real-IP $remote_addr;
    #     proxy_set_header X-Forwarded-For $proxy_add_x_fo
    #     proxy_set_header X-Forwarded-Proto $scheme;
    #     proxy_pass http://jenkins:8080/;
        # proxy_buffering off;
        # proxy_redirect http://jenkins:8080/ https://i11b

        # # Substituting paths for Jenkins static files
        # sub_filter 'href="/userContent' 'href="/jenkins/
        # sub_filter 'src="/userContent' 'src="/jenkins/us
        # sub_filter 'href="/static' 'href="/jenkins/stati
        # sub_filter 'src="/static' 'src="/jenkins/static'
        # sub_filter_once off;
    }
}
```

4. Flask를 이용한 여러 서버 통신

- AWS(gateway 역할) 기준
  - app.py

```
import os
from flask import Flask, request, abort, jsonify
import requests
from queue import Queue
from threading import Thread
from io import BytesIO
import base64

app = Flask(__name__)
response_listener = Flask(__name__)

# 허용할 IP 주소들을 리스트로 정의합니다.
# ALLOWED_IPS = ['175.209.203.185', '43.203.243.195', '110
```

```
# 각 엔드포인트에 대한 타겟 IP를 정의합니다.
TARGET_IPS = {
    'yolo': 'http://110.15.196.166:33333',
    'rag': 'https://magnetic-ram-brave.ngrok-free.app/rag'
    'store': 'http://110.15.196.166:33333',
    'story': 'http://175.209.203.185:33333',
    'creature': 'http://175.209.203.185:33333'  # 요청을 333
}

# GPU 관련 엔드포인트 목록
GPU_ENDPOINTS = ['creature', 'story']

# 요청 큐 생성
request_queue = Queue()
# BACKEND_RESPONSE_URL_TEMPLATE = 'https://i11b104.p.ssafy

def worker():
    while True:
        endpoint, data, callback_url = request_queue.get()
        try:
            if endpoint == 'yolo':
                user_info = data['user_info']
                image_data = BytesIO(data['image'])
                files = {'image': image_data}

                response_yolo = requests.post(f"{TARGET_IP

                if response_yolo.status_code == 200:
                    yolo_response_json = response_yolo.jso

                    # Prepare the GIF file and data to sen
                    image_data.seek(0)
                    encoded_image = base64.b64encode(image_

                    data = {
                        'color': yolo_response_json['color
                        'attribute': yolo_response_json['a
```

```
                    'user_info': yolo_response_json['u
                    'image': encoded_image
                }

                headers = {'Content-Type': 'applicatio
                response_creature = requests.post(f"{T

                ########## # 포트가 달라서 여기 못옴
                # print(f'Worker {response_creature}')
                # # Send GIF to BACKEND_RESPONSE_URL_T
                # # backend_response_url = BACKEND_RES

                # gif_files = {f'image{i}.gif': (f'ima

                # response = requests.post(backend_res
                # print("Backend response status code:
                # print("Backend response text:", resp

                # # Send data and GIF files to store e
                # store_url = f"{TARGET_IPS['store']}/
                # response_store = requests.post(store

                # print("Store response status code:",
                # print("Store response text:", respon


            elif endpoint == 'rag':
                response = requests.post(f"{TARGET_IPS['ra

            elif endpoint == 'store':
                response = requests.post(f"{TARGET_IPS['st

            # 응답을 콜백 URL로 전달 (GPU 서버가 아닌 경우)
            if endpoint not in GPU_ENDPOINTS and endpoint
                requests.post(callback_url, json=response.

        except Exception as e:
            requests.post(callback_url, json={"error": str
```

```
        finally:
            request_queue.task_done()

# 백그라운드에서 큐를 처리하는 스레드 시작
Thread(target=worker, daemon=True).start()

# @app.before_request
# def limit_remote_addr():
#     if request.remote_addr not in ALLOWED_IPS:
#         abort(403)  # Forbidden, 접근 거부

@app.route('/<endpoint>', methods=['POST'])
def enqueue_request(endpoint):
    if endpoint not in TARGET_IPS:
        abort(404)

    # 콜백 URL 설정: GPU 서버의 경우 37777 포트로 설정
    if endpoint in GPU_ENDPOINTS or endpoint == 'yolo':  #
        callback_url = 'http://127.0.0.1:37777/api/{}/resp
    else:
        callback_url = 'https://i11b104.p.ssafy.io/api/{}/

    if endpoint == 'yolo':
        image = request.files.get('image')
        if not image:
            abort(400, description="No image provided")

        user_info = request.form.to_dict()
        image_data = image.read()  # 이미지 데이터를 미리 읽어
        data = {'image': image_data, 'user_info': user_inf

        # 어차피 여기로 못들어 오고 json 파일로만 바꾸면 됨
    else:
        data = request.get_json()
        if not data:
            abort(400, description="No data provided")

    request_queue.put((endpoint, data, callback_url))
```

```
    return jsonify({"message": f"{endpoint.capitalize()} r


@response_listener.route('/api/<endpoint>/response', metho
def handle_response(endpoint):
    try:
        # 데이터 수신
        data = request.json  # Assuming the data is sent a

        # print(f"Received response for {endpoint}: {data}

        # store_url로 데이터를 전송
        headers = {'Content-Type': 'application/json'}
        store_url = f"{TARGET_IPS['store']}/store"
        response_store = requests.post(store_url, json=dat

        print("Store response status code:", response_stor
        print("Store response text:", response_store.text)

        return jsonify({"status": "received", "data": data

    except Exception as e:
        print(f"Error handling response for {endpoint}: {e
        return jsonify({"error": str(e)}), 500


# 새로운 GET 요청 핸들러 추가
@app.route('/store/creature_diary/<nickname>/<date>', meth
def proxy_creature_diary(nickname, creature_name, date):
    store_url = f"{TARGET_IPS['store']}/store/creature_dia
    response = requests.get(store_url)

    if response.status_code == 200:
        return send_file(BytesIO(response.content), mimety
    else:
        return jsonify({"error": "Diary image not found"})

@app.route('/store/<nickname>/<creature_name>/<level>', me
```

```
def proxy_creature_data(nickname, creature_name, level):
    store_url = f"{TARGET_IPS['store']}/store/{nickname}/{
    response = requests.get(store_url)

    if response.status_code == 200:
        return jsonify(response.json())
    else:
        return jsonify({"error": "Creature data not found"


# 33333 포트에서 요청을 처리하는 메인 서버 실행
if __name__ == '__main__':
    Thread(target=lambda: app.run(host='0.0.0.0', port=333

    # 37777 포트에서 응답을 처리하는 리스너 서버 실행
    response_listener.run(host='0.0.0.0', port=37777)
```

- Dockerfile

```
# Python 이미지 기반으로 시작합니다.
FROM python:3.9-slim

# 작업 디렉토리 설정
WORKDIR /app

# Python 의존성 파일 복사 및 설치
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# 애플리케이션 소스 코드 복사
COPY . .

# 애플리케이션이 시작될 포트를 설정
EXPOSE 33333
EXPOSE 37777

# 애플리케이션 실행
CMD ["python", "app.py"]
```

- requirements.txt

```
Flask==2.0.3
Werkzeug==2.0.3
requests==2.27.1
```

- GPU 서버 기준
  - flask.py

```
import os
from flask import Flask, request, jsonify
from io import BytesIO
import requests
from anything_control_pipeline import AnythingControlPipel

# Set CUDA environment variables
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"] = "1"

# Initialize Flask app
app = Flask(__name__)

# Initialize the model pipeline
model = AnythingControlPipeline()

# Define the target IP for the Creature GPU model
TARGET_IP = 'http://creature_model_ip:port'  # Replace wit

@app.route('/', methods=['POST'])
def handle_request():
    # Extract JSON data from the request
    data = request.json
    color = data['color']
    attribute = data['attribute']
    user_info = data['user_info']

    # Extract the image data from the JSON and convert it
    image_data = BytesIO(data['image'])
```

```
    # Process the image using the model pipeline
    processed_images = model.pipe(color=color, spec=attrib

    # Save the processed images into BytesIO objects
    image_files = []
    for i, img in enumerate(processed_images):
        img_io = BytesIO()
        img.save(img_io, format="GIF")
        img_io.seek(0)
        image_files.append((f"image{i}.gif", img_io))

    # Send the images to the Creature GPU model
    files = {name: (name, file, 'image/gif') for name, fil
    response_creature = requests.post(f"{TARGET_IP}/creatu

    # Return the response from the Creature model along wi
    return jsonify({
        'user_info': user_info,
        'response_creature': response_creature.json()
    })

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=33333)
```

- Local Storage server
  - flask.py

```
from flask import Flask, request, jsonify, abort, send_fil
import os
import json
from datetime import datetime
import requests
from PIL import Image
import base64

app = Flask(__name__)
```

```python
# 데이터가 저장될 기본 경로 설정
BASE_DIR = r'C:\Users\kjg64\OneDrive\바탕 화면\Feedme\store'

# 기본 경로가 없으면 생성
if not os.path.exists(BASE_DIR):
    os.makedirs(BASE_DIR)

@app.route('/store', methods=['POST'])
def store_data():
    """
    데이터 저장 엔드포인트.
    Nickname에 맞게 폴더를 생성하고, 데이터와 GIF 이미지를 저장합니다
    """
    data = request.json
    print(data)

    # 필요한 데이터 확인
    # user_info = data.get('user_info', {})
    nickname = data.get('user_info', {}).get('username')
    creature_name = data.get('user_info', {}).get('creatur
    attribute = data.get('attribute')
    color = data.get('color')
    images = data.get('images', [])  # 여러 개의 이미지 데이터를

    # 데이터가 존재하는지 확인
    if not all([nickname, creature_name, attribute, color,
        abort(400, description="Missing data or files")

    # Nickname에 맞는 폴더 생성
    user_dir = os.path.join(BASE_DIR, nickname)
    if not os.path.exists(user_dir):
        os.makedirs(user_dir)

    # 크리처 폴더 생성
    creature_dir = os.path.join(user_dir, 'creature')
    if not os.path.exists(creature_dir):
        os.makedirs(creature_dir)
```

```
    # 크리처 이름이 None이 아닌지 확인
    if not creature_name:
        abort(400, description="creature_name is missing")

    # 크리처 이름에 맞는 서브 폴더 생성
    creature_subdir = os.path.join(creature_dir, creature_
    if not os.path.exists(creature_subdir):
        os.makedirs(creature_subdir)

    # 데이터 저장
    data_file_path = os.path.join(creature_subdir, 'data.j
    creature_data = {
        "creature_name": creature_name,
        "attribute": attribute,
        "color": color
    }

    with open(data_file_path, 'w') as data_file:
        json.dump(creature_data, data_file)

    # 여러 이미지를 저장
    for index, image_info in enumerate(images):
        file_name = image_info.get('file_name')
        encoded_image = image_info.get('encoded_image')

        if not file_name or not encoded_image:
            print(f"Missing file name or encoded image for
            continue

        try:
            # Base64 디코딩 및 이미지 처리
            image_data_bytes = base64.b64decode(encoded_im
            image_file_path = os.path.join(creature_subdir

            # 이미지를 파일로 저장
            with open(image_file_path, 'wb') as image_file
                image_file.write(image_data_bytes)
```

```
            # 이미지 변환 및 저장 (만약 필요할 경우)
            image = Image.open(image_file_path)
            gif_file_path = os.path.join(creature_subdir,
            image.save(gif_file_path, format='GIF')
        except Exception as e:
            print(f"Failed to process image {index + 1}: {
            abort(500, description=f"Image {index + 1} pro

    return jsonify({"message": "Data and files stored succ

@app.route('/store/creature_diary', methods=['POST'])
def store_creature_diary():
    """
    그림일기 저장 엔드포인트.
    Nickname에 맞게 폴더를 생성하고, 날짜별로 그림일기를 저장합니다.
    """
    data = request.form
    image = request.files.get('image')

    response = requests.post(
        "http://localhost:5000/generate_comics",
        headers={"Content-Type": "application/json"},
        data=json.dumps(data)
    )



    img_base64 = response.json().get('image')
    diary_image = base64.b64decode(img_base64)

    # 필요한 데이터 확인
    nickname = data.get('nickname')

    if not all([nickname, diary_image]):
        abort(400, description="Missing data or diary imag

    # Nickname에 맞는 폴더 생성
    user_dir = os.path.join(BASE_DIR, nickname)
    if not os.path.exists(user_dir):
```

```
        os.makedirs(user_dir)

    # 그림일기 폴더 생성
    diary_dir = os.path.join(user_dir, 'creature_diary')
    if not os.path.exists(diary_dir):
        os.makedirs(diary_dir)

    # 날짜별로 폴더를 생성하여 그림일기를 저장하는 대신, 이미지 파일명(
    date_str = datetime.now().strftime('%Y-%m-%d')
    diary_image_path = os.path.join(diary_dir, f'{date_str
    diary_image.save(diary_image_path)

    return jsonify({"message": "Creature diary stored succ


@app.route('/store/creature_diary/<nickname>/<date>', meth
def retrieve_creature_diary(nickname, date):
    """
    그림일기 조회 엔드포인트.
    Nickname, 크리처 이름, 날짜를 기반으로 그림일기를 조회합니다.
    """
    # 크리처 네임 필요 없음
    diary_image_path = os.path.join(BASE_DIR, nickname, 'c

    if not os.path.exists(diary_image_path):
        abort(404, description="Diary image not found")

    return send_file(diary_image_path, mimetype='image/jpe

@app.route('/store/<nickname>/<creature_name>/<level>', me
def retrieve_creature_data(nickname, creature_name, level)
    """
    크리처 데이터 조회 엔드포인트.
    Nickname, 크리처 이름, Level을 기반으로 데이터를 조회합니다.
    """
    # 크리처 폴더 경로
    creature_subdir = os.path.join(BASE_DIR, nickname, 'cr
```

```
    if not os.path.exists(creature_subdir):
        abort(404, description="Creature data not found")

    # 데이터 파일 경로
    data_file_path = os.path.join(creature_subdir, 'data.j
    if not os.path.exists(data_file_path):
        abort(404, description="Data file not found")

    with open(data_file_path, 'r') as data_file:
        creature_data = json.load(data_file)

    # level 값을 이용한 GIF 파일 경로
    gif_file_path = os.path.join(creature_subdir, f'{level

    if os.path.exists(gif_file_path):
        # GIF 파일을 읽고 base64로 인코딩
        with open(gif_file_path, "rb") as gif_file:
            gif_data = gif_file.read()
            encoded_gif = base64.b64encode(gif_data).decod

        # JSON 응답 생성
        response_data = {
            "nickname": nickname,
            "creature_name": creature_name,
            "level": level,
            "gif_data": encoded_gif  # base64로 인코딩된 GIF
        }

        return jsonify(response_data)
    else:
        abort(404, description=f"{level}.gif file not foun


@app.route('/yolo', methods=['POST'])
def handle_yolo_request():
    """
    YOLO 요청을 받아 localhost 37777로 보내고 그 결과를 반환합니다
    """
```

```
    # 받은 데이터를 그대로 전송
    response = requests.post(" http://192.168.35.215:37777

    if response.status_code == 200:
        return jsonify(response.json())
    else:
        return abort(response.status_code)

@app.route('/store/egg/<level>', methods=['GET'])
def retrieve_egg_data(level):
    """
    알 데이터 조회 엔드포인트.
    Level을 기반으로 데이터를 조회합니다.
    """
    # 크리처 폴더 경로
    creature_subdir = os.path.join(BASE_DIR, 'egg', 'creat

    if not os.path.exists(creature_subdir):
        abort(404, description="Creature data not found")

    # 데이터 파일 경로
    data_file_path = os.path.join(creature_subdir, 'data.j
    if not os.path.exists(data_file_path):
        abort(404, description="Data file not found")

    with open(data_file_path, 'r') as data_file:
        creature_data = json.load(data_file)

    # level 값을 이용한 GIF 파일 경로
    gif_file_path = os.path.join(creature_subdir, f'{level

    if os.path.exists(gif_file_path):
        # GIF 파일을 읽고 base64로 인코딩
        with open(gif_file_path, "rb") as gif_file:
            gif_data = gif_file.read()
            encoded_gif = base64.b64encode(gif_data).decod

        # JSON 응답 생성
```

```
            response_data = {
                "gif_data": encoded_gif  # base64로 인코딩된 GIF
            }

            return jsonify(response_data)
        else:
            abort(404, description=f"{level}.gif file not foun

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=33333)
```

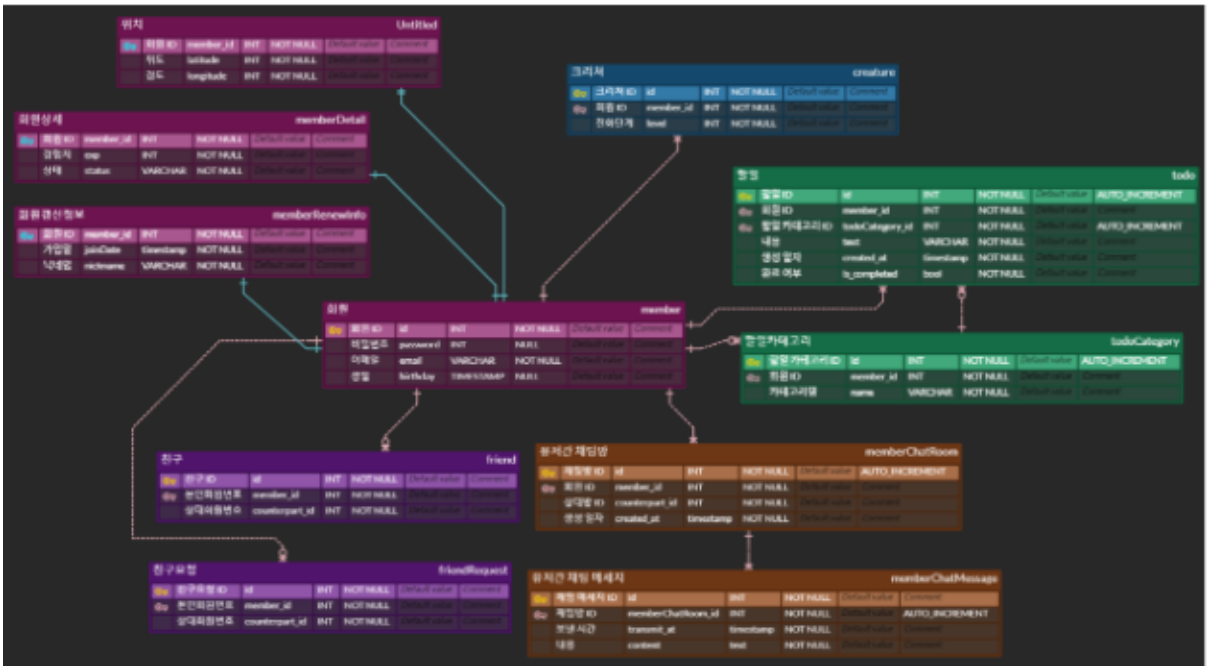
- yolo_flask.py
- stroy_diffusion.py

> 특이사항
> - 각각의 server별로 flask와 port를 구축해줘야 한다
> - 최소 6개정도의 flask 구현이 필요
> - 각각의 api_key 필요

5. 외부 서비스 정리

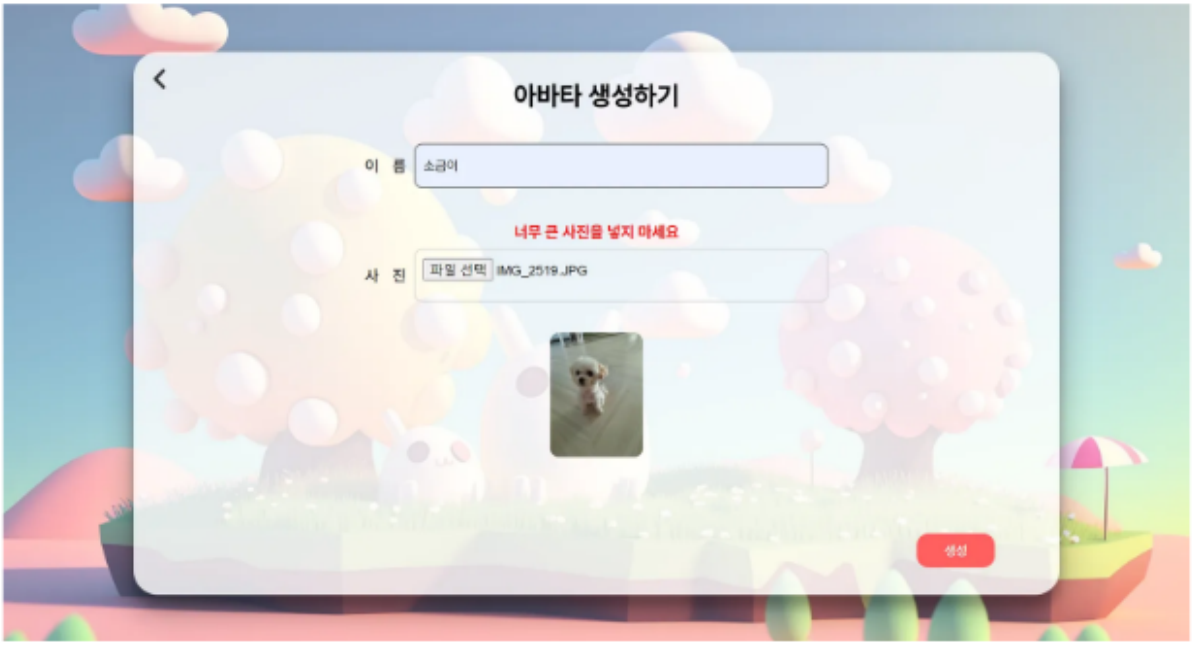
```
## 소셜인증
- Naver
- kakao
## 배포
- Gitlab
- jenkins
- github
- Figma
- weather api
```

6. DB ERD

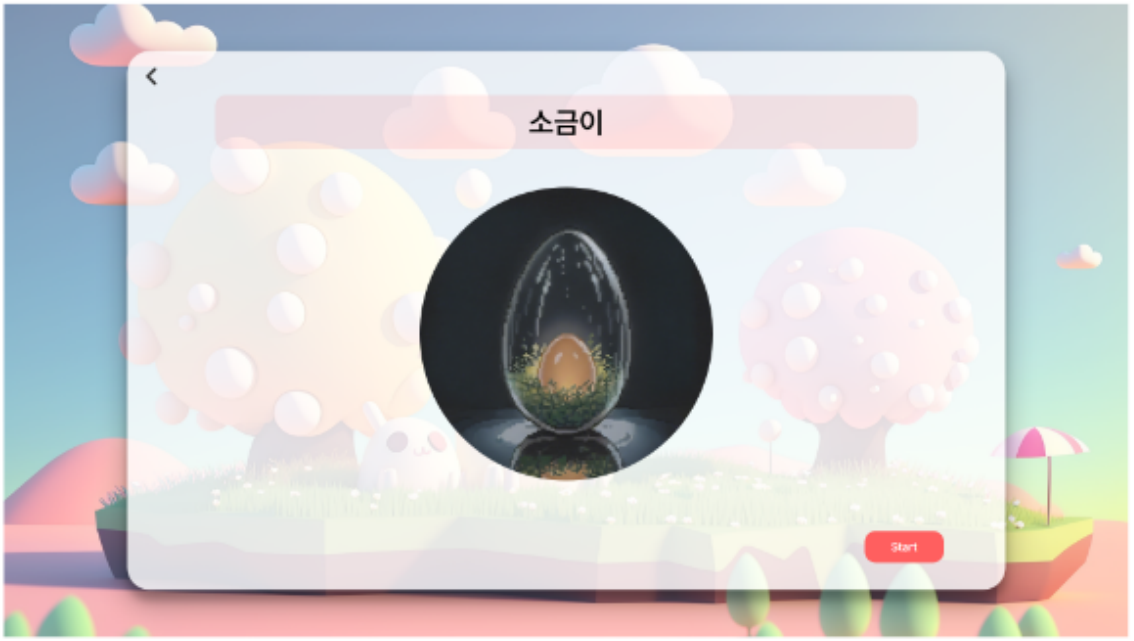


7. 서비스 로직

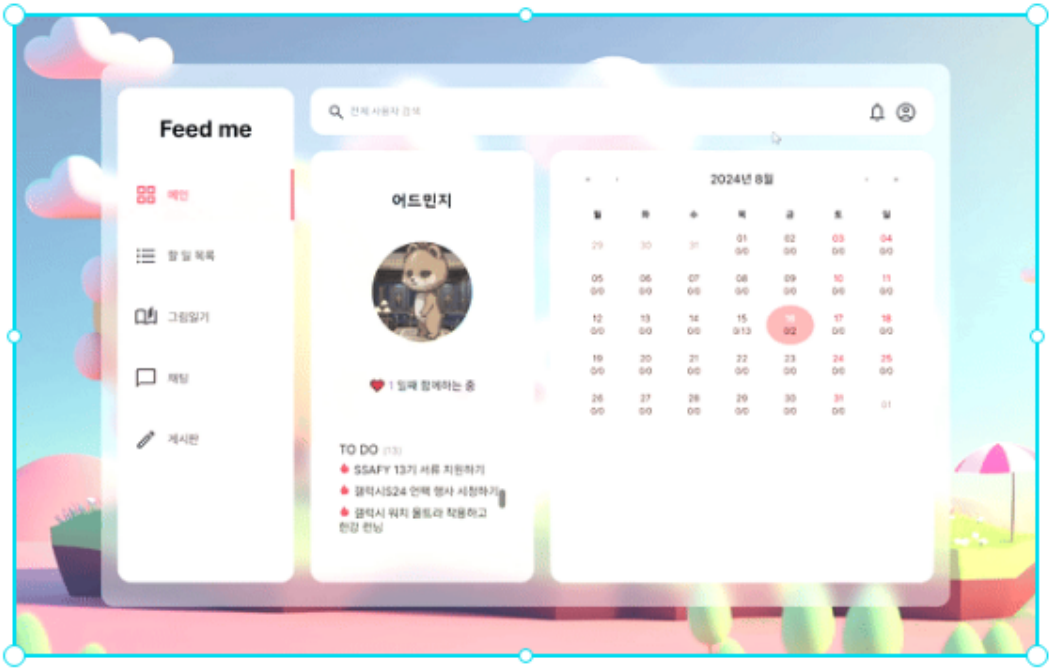
1. 소셜로그인을 통한 회원가입 및 내가 만들고 싶은 사진을 이용한 아바타 생성



2. 알 이미지를 띄어줌

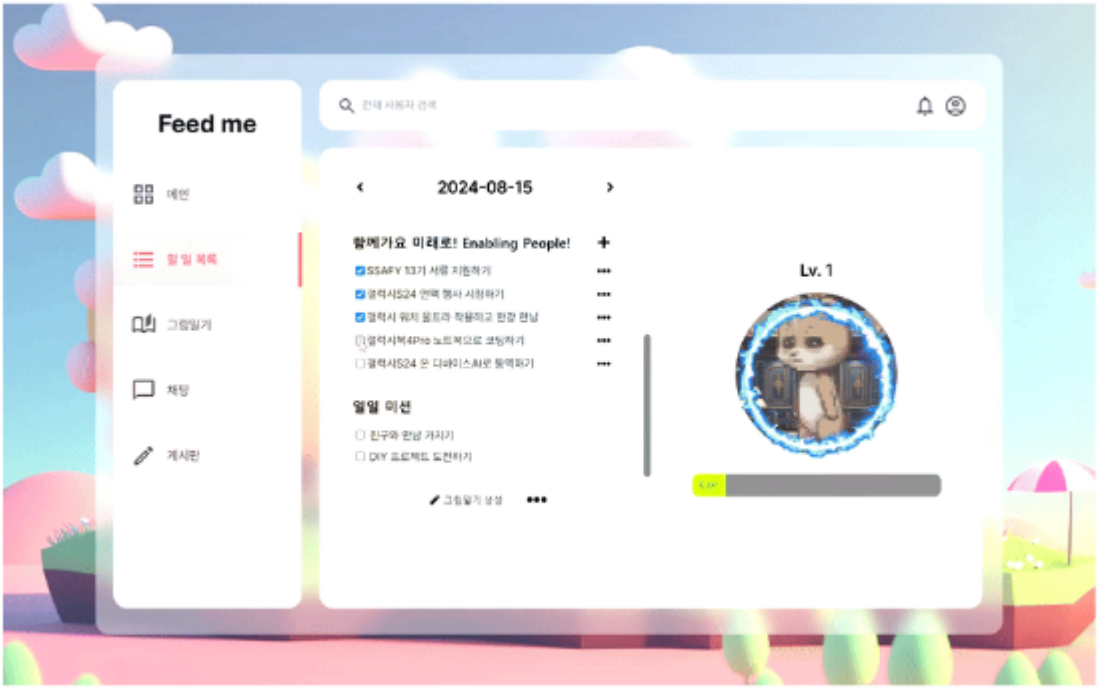


3. 아바타 생성 후의 메인 화면 - 날씨, 위치 기반의 크리처 메인화면 Effect 다름

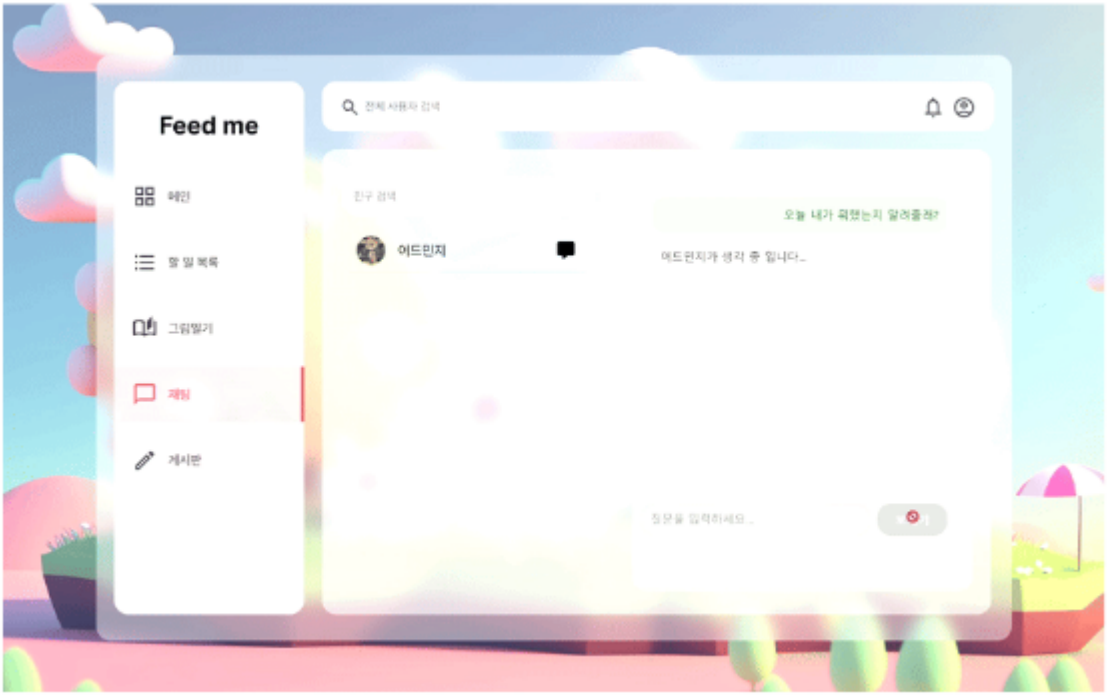


4. Open weather API - 날씨, 위치기반 일일 미션 제공

5. 할일 완료를 통한 Effect, 크리처는 (GIF 이미지)



6. Rag 바탕의 개인화된 채팅 구현



7. Data 서버 바탕의 이미지 구현