Ama Sefah
KDG

[KG1] Endpoint Definitions:

- File Reference: post_service\main.py lines 78 - 91
- Code Fragment:

```python
@app.post("/posts", status_code=201, response_model=PostResponse)
async def create_post(post: PostCreate):

    user_req = f"{USER_SERVICE_BASE}/users/{post.user_id}"
    user = httpx.get(user_req)

    if not user.status_code == 200:
        raise HTTPException(status_code=404, detail= f"User {post.user_id} not found!")

    with get_session() as session:
        new_post = create_new_post(session, post)

        logger.info(f"Post {new_post['post_id']} created by user {post.user_id}")
        return new_post
```

- Justification: This is a well defined endpoint for users to create a post. It takes in a pydantic model and returns the correct response to the user.

[KG2] Containerization:

- File Reference: comment_service\Dockerfile
- Code Fragment:

```
comment_service > Dockerfile > ...
1   FROM python:3.11-slim
2
3   WORKDIR /33646991_CapstoneProject
4
5   RUN apt-get update && apt-get install -y curl
6
7   COPY requirements.txt .
8   RUN pip install --no-cache-dir -r requirements.txt
9
10  COPY . .
11
12  CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

- This dockerfile defines how to build comment_service and where the work directory is. The application after set up is then bundled into a single container for just the comment service which is containerization.

[KG3] Container Orchestration:

- File Reference: docker-compose.yml
- Code Fragment:

-
```yaml
user_service:
  build:
    context: ./user_service
    dockerfile: Dockerfile
  container_name: user_service
  ports:
    - "8000:8000"
  environment:
    - DATABASE_URL=postgresql+psycopg2://blogspotuser-table:blogspotuserpass@user_db:5432/blogspot
  depends_on:
    user_db:
      condition: service_healthy
  networks:
    - blogspot-network
  restart: unless-stopped
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
```

- This docker-compose file defines all the services and manages them. This is an example of container orchestration because it is managing all the containers and making sure that the correct dependencies are good before startup.

[KG4] API Gateway:

- File Reference: docker-compose.yml
- Code Fragments:

-
```yaml
nginx:
  image: nginx:latest
  ports:
    - "80:80"
  volumes:
    - ./nginx.conf:/etc/nginx/conf.d/default.conf
  networks:
    - blogspot-network
  depends_on:
    - user_service
    - post_service
    - comment_service
    - trending_service
  restart: unless-stopped
```

- This is a representation of API Gateway because it exposes port 80 to the public and reroutes any traffic back to its accurate service. Because it is running all the services, it depends on them before it can startup.

[KG5] Load Balancing:

- File Reference: nginx.conf
- Code Fragments:

-
```
    upstream comment_service {
        server comment_service:8002;
    }

    upstream trending_service {
        server trending_service:8003;
    }
```

- This is an example of load balancer because it is using nginx to set servers and ports for the application.

[KG6] Reliability, Scalability, Maintainability:

- File Reference: Docker-compose.yml
- Code Fragments:
-
```
      - DATABASE_URL=postgresql+psycopg2://blogspotuser-table:blogspotuserpass@user_db:5432/blogspo
    depends_on:
      user_db:
        condition: service_healthy
    networks:
```

- This demonstrates reliability because it shows that the services lean on each other to run so if one part has an error, it will be discovered quickly and intervention can occur.

[KG7] Service Communication:

- File Reference: trending_service\main.py

- Code Fragments:
  -

```
async with httpx.AsyncClient(timeout=2.0) as client:
    user_resp = await client.get(f"{USER_SERVICE_BASE}/health")
if user_resp.status_code != 200:
    return {
        "status": "unhealthy",
        "details": "User Service returned non-200",
        "code": user_resp.status_code
    }

async with httpx.AsyncClient(timeout=2.0) as client:
    post_resp = await client.get(f"{POST_SERVICE_BASE}/health")

if post_resp.status_code != 200:
    return {
        "status": "unhealthy",
        "details": "Post Service returned non-200",
        "code": post_resp.status_code
    }
```

- This demonstrates service communication because the trending service is contacting both the user service and the post service to make sure it is up and running before reporting as healthy.

[KG10] Database Per Service Pattern:

- File Reference: comment_service\comments.sql
- Code Fragments:

-

comment_service > comments.sql

```sql
CREATE TABLE IF NOT EXISTS comments (
    comment_id   SERIAL PRIMARY KEY,
    post_id      VARCHAR NOT NULL,
    user_id      VARCHAR NOT NULL,
    username     VARCHAR(50) NOT NULL UNIQUE,
    content      VARCHAR(500) NOT NULL,
    likes        INTEGER NOT NULL DEFAULT 0,
    dislikes     INTEGER NOT NULL DEFAULT 0
    edited_at    TIMESTAMP NOT NULL DEFAULT NOW()
);

```

- This showcases ‘database per set’ because it is an sql file that defines the schema for only the comments service. The comments service only has access to this database.