Group CO.SDH3-A

Rastvorov Vladyslav

Student ID R00274535

# **HR System Application**

This is a distributed HR system application implemented with gRPC, RabbitMQ, and Docker. It includes a client, server, and RabbitMQ consumer service to process employee-related queries and logs system activities.

## **Features**

1. Employee Query System:

* Retrieve employee salary details.
* Fetch vacation leave entitlement and history.
* Handle unrecognized employee IDs and invalid queries gracefully.

1. Activity Logging:

* Logs all employee queries to a RabbitMQ message queue.
* RabbitMQ consumer processes and stores the logs.

1. Multi-client Support:

* The server supports multiple simultaneous clients using gRPC.

1. Dockerized Environment:

* Fully containerized using Docker Compose.

## **System Architecture**

1. RabbitMQ:

* Acts as the message broker to handle activity logs.

1. gRPC Server:

* Processes employee queries from clients.
* Publishes logs to RabbitMQ.

1. Client:

* Menu-driven console application for querying the server.

1. RabbitMQ Consumer:

* Consumes and processes activity logs from RabbitMQ.

## **Prerequisites**

* Docker
* Docker Compose

## **Project Structure**

Assignment2/

├── client/ *# Client application*

│ ├── logs/ *# Log directory for client logs*

│ │ └── client.log *# Log file for client operations*

│ ├── client.py *# Client implementation*

│ ├── Dockerfile *# Dockerfile for the client*

│ ├── requirements.txt *# Python dependencies for the client*

│ └── wait-for-server.sh *# Shell script to wait for the gRPC server to start*

├── rabbit\_consumer/ *# RabbitMQ consumer application*

│ ├── logs/ *# Log directory for consumer logs*

│ │ └── consumer.log *# Log file for consumer operations*

│ ├── consumer.py *# RabbitMQ consumer implementation*

│ ├── Dockerfile *# Dockerfile for the RabbitMQ consumer*

│ ├── requirements.txt *# Python dependencies for the RabbitMQ consumer*

│ └── wait-for-rabbitmq-for-consumer.sh *# Shell script to wait for RabbitMQ to start*

├── proto/ *# Protocol Buffer definitions*

│ ├── employee.proto *# Protocol Buffer schema for employee service*

├── server/ *# gRPC server application*

│ ├── logs/ *# Log directory for server logs*

│ │ └── server.log *# Log file for server operations*

│ ├── employee\_pb2\_grpc.py *# Auto-generated gRPC classes*

│ ├── employee\_pb2.py *# Auto-generated Protocol Buffer classes*

│ ├── server.py *# gRPC server implementation*

│ ├── Dockerfile *# Dockerfile for the server*

│ ├── requirements.txt *# Python dependencies for the server*

│ └── wait-for-rabbitmq.sh *# Shell script to wait for RabbitMQ to start*

├── docker-compose.yml *# Docker Compose file to orchestrate services*

├── .gitignore *# Git ignore file*

└── README.md *# Project documentation*

## **Setup and Running the Application**

1. Clone the repository:

git clone <repository\_url>

cd <repository\_name>

1. Build and run the services:

docker-compose up --build

1. Access RabbitMQ Management Interface:

* URL: http://localhost:15672
* Default credentials: guest/guest

## **Testing the Application**

1. When all services are up, the client application runs automatically and prompts for inputs.
2. Follow the on-screen menu to query employee details:

* Enter valid employee IDs to fetch salary or leave details.
* Test with invalid employee IDs or queries to observe error handling.

## **Notes**

1. **Hardcoded Client Responses**:  
   To save time and ensure functionality, the client (client.py) has hardcoded inputs to simulate the assignment's predefined scenario. This provides a seamless demonstration of the expected behavior as outlined in the task description.
2. **Logging**:  
   Each component (client, server, and consumer) generates logs in its respective logs/ directory for easier debugging and tracking of system activity.
3. **Message Queue (RabbitMQ)**:  
   RabbitMQ is used to handle logging of requests and responses, ensuring reliable message delivery.

## **Summary**

This project is a fully containerized application that:

1. Provides employee details through a gRPC server.
2. Logs activity using RabbitMQ.
3. Consumes logs with a RabbitMQ consumer.
4. Simulates client-server interactions using predefined inputs.