# Midterm Design Challenge: Communication Layer

Author: Judrianne Mahigne

NJIT Email: jm2489@njit.edu

This is the Communication Layer description for the fall 2024 IT490 - Systems Integration midterm design project.

## Table of Contents



## Overview

The communication layer controls how the front-end application and data source layer interact. By managing incoming client requests, processing data, and providing the required information in response, it makes data transmission process more efficient.

1. **Message Exchange Server**
    - Manages incoming and outgoing messages between the client and the back end.
    - Uses a message broker (RabbitMQ) to route messages to respected queues.
2. **Queues**
    - Organize and manage requests for different functions of the application.
3. **Request Handlers**
    - Define functions that process client requests, interact with the data source layer, and return results.
4. **Consumer Processes**
    - Processes that continuously listen to queues, fetch data, and return results to the client.

## Implementation

### 1. Message Exchange System

- **Architecture**: Use a message broker like **RabbitMQ** to manage communication between the front end and back end.
- **Exchange Types**:
    - **Direct Exchange**: For routing messages to a specific queue based on routing keys.

### 2. Queues

- **Queue Setup**:
  - **ParticipantsQueue**:
    - Responds to incoming queries for participant data (such as the infection status or participant list).
    - Directs queries to functions that return participant data and communicate with the API endpoints.
  - **LocationsQueue**:
    - Oversees participant location requests, and geo information for map visualization.
    - Results are returned after processing queries to the `getParticipantLocations` API for example.
  - **TrendsQueue**:
    - For trend analysis, infection data is collected over certain time periods.
    - Data is generated for graphs and requests are processed by the data source layer.
  - **ProjectionsQueue**:
    - Responds to queries for infection projection based on data patterns.
    - Calculations are started in the data source layer, and future outcomes are returned.

## 3. Functions Available to the Client

The communication layer would use the following functions for front end communication:

- **getListOfParticipants()**:
  - **Description**: Retrieves a list of participants along with their infection status.
  - **Request Type**: Direct exchange, routed to **ParticipantsQueue**.
  - **Example Response**:

```
[
  { "id": 123, "age": 35, "infected": "2024-09-15T12:00:00Z",
"recovered": false },
  { "id": 124, "age": 28, "infected": null, "Recovered": null }
]
```

- **getLocations(id)**:
  - **Description**: Gets the GPS coordinates and timestamps for a specific participant.
  - **Request Type**: Direct exchange, routed to **LocationsQueue**.
  - **Example Response**:

```
[
  { "id": 1, "latitude": 40.7128, "longitude": -74.0060,
"created_at": "2024-10-25T14:00:00Z" },
  { "id": 2, "latitude": 40.7306, "longitude": -73.9352,
"created_at": "2024-10-25T15:30:00Z" }
]
```

- **getInfectionTrends(zipCode, startTime, endTime)**:
  - **Description**: Retrieves collected infection counts by zip code over a given time range.
  - **Request Type**: Direct exchange, routed to **TrendsQueue**.
  - **Example Response**:

```
{ "zip": "07410", "infected": 100, "startTime": "2024-10-01",
"endTime": "2024-10-31" }
```

### 4. Consumer Processes

- **Consumers** continuously listen to their respective queues, process incoming messages, and communicate with the data source layer to fetch, aggregate, or transform data.
- **Implementation**:
  - Consumers validate incoming data, interact with APIs or databases, and send responses back to the front end.
  - Example code for a consumer in `python`:

```python
import pika

def process_message(ch, method, properties, body):
    data = json.loads(body)
    result = fetch_participant_data(data['participantId'])
    send_response(result)

# Setup RabbitMQ consumer
connection =
pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='ParticipantsQueue')
channel.basic_consume(queue='ParticipantsQueue',
on_message_callback=process_message, auto_ack=True)

print('Waiting for messages...')
channel.start_consuming()
```

## Security

- **Secure Messaging**: Use TLS to encrypt communication between the client and the message broker.
- **Implement Retry Mechanisms**: Ensure that messages are reprocessed in case of failures.
- **Use Dead Letter Queues**: To capture unprocessable messages for further analysis and debugging.
- **Monitor Queue Health**: Regularly check the state of the queues to ensure smooth operation and identify potential bottlenecks.

## Technologies

- **Message Broker**: RabbitMQ or similar messaging queing system

- **Programming Language**: Python (pika library)
- **Monitoring Tools**: RabbitMQ Management Plugin