Team Number: 23
Project Title: Message Broker
Report Date: 1/28/2024
Part Four: Progress status as a team

1. Have you met as a team this past week? Yes, February 2nd 3:30pm with Quang Le, Maximo Mejia, Mahmood Sakib, Aaron Steig, Kyla Tolentino

2. Have you met with the sponsor as a team? Yes, January 31st 4:00pm with Quang Le, Maximo Mejia, Mahmood Sakib, Aaron Steig, Kyla Tolentino

3. Describe verbally the tasks completed the past week and the challenges faced:
Throughout the week, our focus was on various assignments related to Docker and Rabbit MQ. Primarily, we conducted quality assurance testing for both our message broker and parser module, while also delving into research on Flask integration.

4. Describe the tasks to be completed the coming week.
Encourage collective teamwork for evaluating the current project status and assigning roles, thereby ensuring the project adheres to the established schedule. This involves assigning distinct responsibilities to team members, including tasks such as documentation, project status evaluation, test planning, and allocation of responsibilities for upcoming stages of project development.

Previous Report Summary. Date: 1/28/2024

	# Tasks completed past week	**# Tasks not completed**	**# Tasks for next week**
Quang Le	2	0	2
Maximo Mejia	2	0	2
Mahmood Sakib	2	0	2
Aaron Steig	2	0	2
Kyla Tolentino	2	0	2

Current Report Summary. Date: 2/4/2024

	# Tasks completed past week	**# Tasks not completed**	**# Tasks for next week**
Quang Le	2	0	2
Maximo Mejia	2	0	2
Mahmood Sakib	2	0	2
Aaron Steig	2	0	2
Kyla Tolentino	2	0	2

Name of the Member: Quang Le
Report Date: 1/28/2024

Previous:

1. Final developments on Docker environment for system testing (01/28/2024)

Review the Docker file for any necessary final adjustments to ensure optimal functionality. Perform last-minute routine checks to deploy the project in the advanced Docker environment. Present a comprehensive report to the team upon completion.

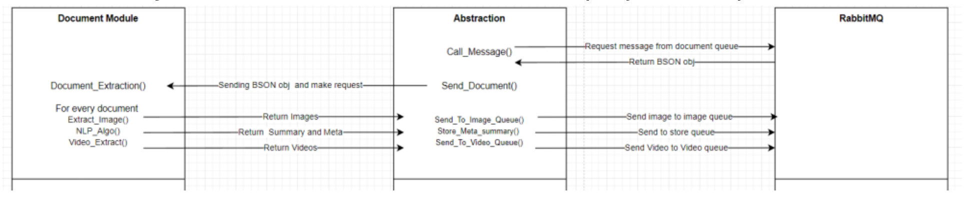
2. Research Flask integration (01/25/2024)

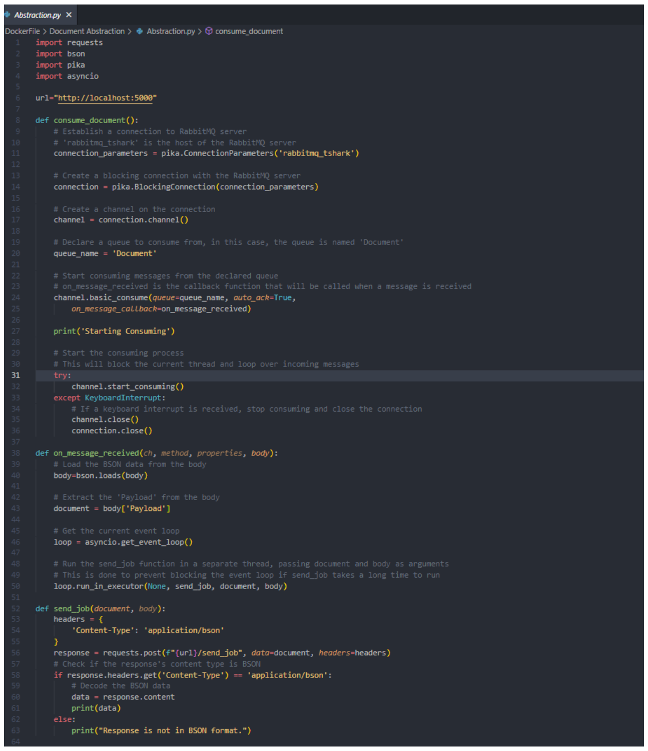
Clarifying the operational aspects of Flask integration and offering guidance on incorporating this knowledge into our project, encompassing integration with databases, third-party libraries, and diverse technologies.

Done:

3. Initial implementation of the abstraction layer

Concentrating on transforming abstract concepts into tangible code and structures that act as a link between various components of the software system. Verify that the design harmonizes with the overall system architecture and fulfills the project's requirements.

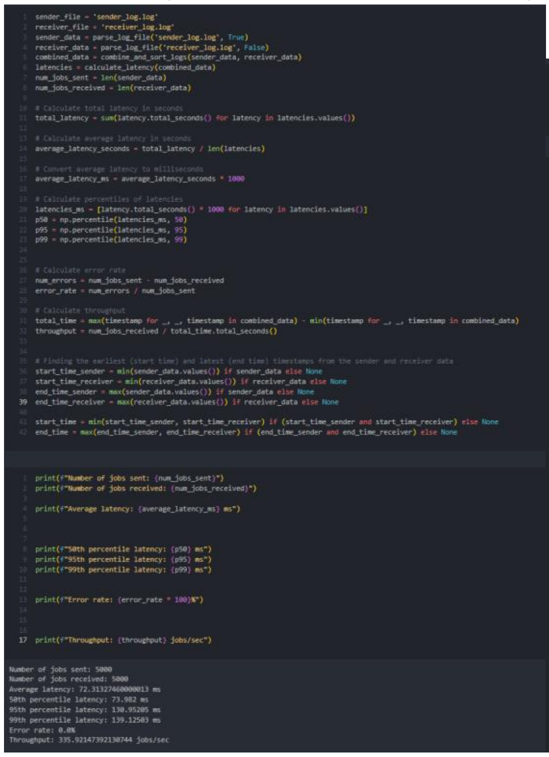



Abstraction.py
DockerFile > Document Abstraction > Abstraction.py > consume_document

```python
import requests
import bson
import pika
import asyncio

url="http://localhost:5000"

def consume_document():
    # Establish a connection to RabbitMQ server
    # 'rabbitmq_tshark' is the host of the RabbitMQ server
    connection_parameters = pika.ConnectionParameters('rabbitmq_tshark')

    # Create a blocking connection with the RabbitMQ server
    connection = pika.BlockingConnection(connection_parameters)

    # Create a channel on the connection
    channel = connection.channel()

    # Declare a queue to consume from, in this case, the queue is named 'Document'
    queue_name = 'Document'

    # Start consuming messages from the declared queue
    # on_message_received is the callback function that will be called when a message is received
    channel.basic_consume(queue=queue_name, auto_ack=True,
        on_message_callback=on_message_received)

    print('Starting Consuming')

    # Start the consuming process
    # This will block the current thread and loop over incoming messages
    try:
        channel.start_consuming()
    except KeyboardInterrupt:
        # If a keyboard interrupt is received, stop consuming and close the connection
        channel.close()
        connection.close()

def on_message_received(ch, method, properties, body):
    # Load the BSON data from the body
    body=bson.loads(body)

    # Extract the 'Payload' from the body
    document = body['Payload']

    # Get the current event loop
    loop = asyncio.get_event_loop()

    # Run the send_job function in a separate thread, passing document and body as arguments
    # This is done to prevent blocking the event loop if send_job takes a long time to run
    loop.run_in_executor(None, send_job, document, body)

def send_job(document, body):
    headers = {
        'Content-Type': 'application/bson'
    }
    response = requests.post(f"{url}/send_job", data=document, headers=headers)
    # Check if the response's content type is BSON
    if response.headers.get('Content-Type') == 'application/bson':
        # Decode the BSON data
        data = response.content
        print(data)
    else:
        print("Response is not in BSON format.")
```

4. Make adjustments based on the QA report

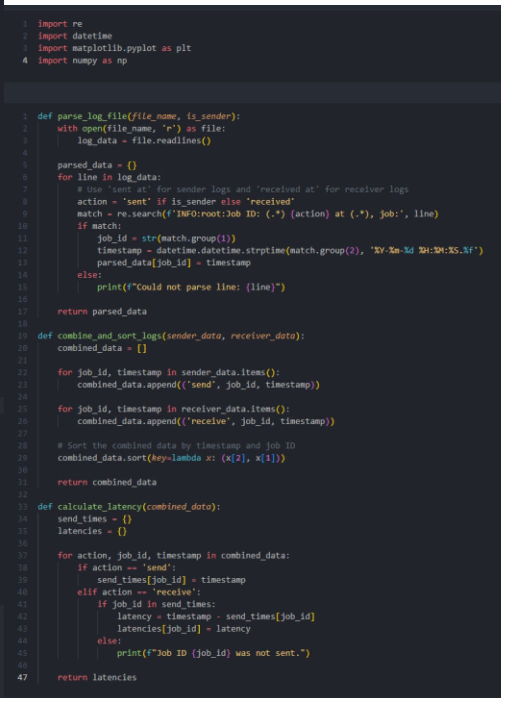
Based on the notable findings and observations from the testing, I made refined adjustments that are necessary to enhance the functionality of the message broker.



```python
sender_file = 'sender_log.log'
receiver_file = 'receiver_log.log'
sender_data = parse_log_file('sender_log.log', True)
receiver_data = parse_log_file('receiver_log.log', False)
combined_data = combine_and_sort_logs(sender_data, receiver_data)
latencies = calculate_latency(combined_data)
num_jobs_sent = len(sender_data)
num_jobs_received = len(receiver_data)

# Calculate total latency in seconds
total_latency = sum(latency.total_seconds() for latency in latencies.values())

# Calculate average latency in seconds
average_latency_seconds = total_latency / len(latencies)

# Convert average latency to milliseconds
average_latency_ms = average_latency_seconds * 1000

# Calculate percentiles of latencies
latencies_ms = [latency.total_seconds() * 1000 for latency in latencies.values()]
p50 = np.percentile(latencies_ms, 50)
p95 = np.percentile(latencies_ms, 95)
p99 = np.percentile(latencies_ms, 99)


# Calculate error rate
num_errors = num_jobs_sent - num_jobs_received
error_rate = num_errors / num_jobs_sent

# Calculate throughput
total_time = max(timestamp for _, _, timestamp in combined_data) - min(timestamp for _, _, timestamp in combined_data)
throughput = num_jobs_received / total_time.total_seconds()


# Finding the earliest (start time) and latest (end time) timestamps from the sender and receiver data
start_time_sender = min(sender_data.values()) if sender_data else None
start_time_receiver = min(receiver_data.values()) if receiver_data else None
end_time_sender = max(sender_data.values()) if sender_data else None
end_time_receiver = max(receiver_data.values()) if receiver_data else None

start_time = min(start_time_sender, start_time_receiver) if (start_time_sender and start_time_receiver) else None
end_time = max(end_time_sender, end_time_receiver) if (end_time_sender and end_time_receiver) else None
```

```python
print(f"Number of jobs sent: {num_jobs_sent}")
print(f"Number of jobs received: {num_jobs_received}")

print(f"Average latency: {average_latency_ms} ms")



print(f"50th percentile latency: {p50} ms")
print(f"95th percentile latency: {p95} ms")
print(f"99th percentile latency: {p99} ms")


print(f"Error rate: {error_rate * 100}%")



print(f"Throughput: {throughput} jobs/sec")
```

```
Number of jobs sent: 5000
Number of jobs received: 5000
Average latency: 72.31327460000013 ms
50th percentile latency: 73.982 ms
95th percentile latency: 130.95205 ms
99th percentile latency: 139.12503 ms
Error rate: 0.0%
Throughput: 335.92147392130744 jobs/sec
```



```python
import re
import datetime
import matplotlib.pyplot as plt
import numpy as np
```

```python
def parse_log_file(file_name, is_sender):
    with open(file_name, 'r') as file:
        log_data = file.readlines()

    parsed_data = {}
    for line in log_data:
        # Use 'sent at' for sender logs and 'received at' for receiver logs
        action = 'sent' if is_sender else 'received'
        match = re.search(f'INFO:root:Job ID: (.*) {action} at (.*), job:', line)
        if match:
            job_id = str(match.group(1))
            timestamp = datetime.datetime.strptime(match.group(2), '%Y-%m-%d %H:%M:%S.%f')
            parsed_data[job_id] = timestamp
        else:
            print(f"Could not parse line: {line}")

    return parsed_data

def combine_and_sort_logs(sender_data, receiver_data):
    combined_data = []

    for job_id, timestamp in sender_data.items():
        combined_data.append(('send', job_id, timestamp))

    for job_id, timestamp in receiver_data.items():
        combined_data.append(('receive', job_id, timestamp))

    # Sort the combined data by timestamp and job ID
    combined_data.sort(key=lambda x: (x[2], x[1]))

    return combined_data

def calculate_latency(combined_data):
    send_times = {}
    latencies = {}

    for action, job_id, timestamp in combined_data:
        if action == 'send':
            send_times[job_id] = timestamp
        elif action == 'receive':
            if job_id in send_times:
                latency = timestamp - send_times[job_id]
                latencies[job_id] = latency
            else:
                print(f"Job ID {job_id} was not sent.")

    return latencies
```

In-Progress:

5. Assist in the development of API for abstraction layer

Actively collaborating to facilitate the development of a specialized Application Programming Interface tailored for the abstraction layer. This included actively participating in both the creation and enhancement of the interface, contributing significantly to its functionality and effectiveness within the broader system architecture.

6. Perform fixes to the parser based on the QA report

Conducting a comprehensive analysis of the Quality Assurance (QA) report and implementing meticulous fixes to enhance the functionality, accuracy, and alignment with quality standards of the parser module. The iterative process included a thorough examination of reported issues, strategic planning for fixes, and rigorous testing to ensure the successful resolution of each identified concern.

Name of the Member: Maximo Mejia
Report Date: 1/28/2024

Previous:

1. Develop QA testing for message broker (01/25/2024)

Conduct comprehensive research and iterative testing to formulate customized QA testing plans tailored for RabbitMQ. Prioritize the evaluation of the message broker's reliability, functionality, and performance throughout the testing procedure.

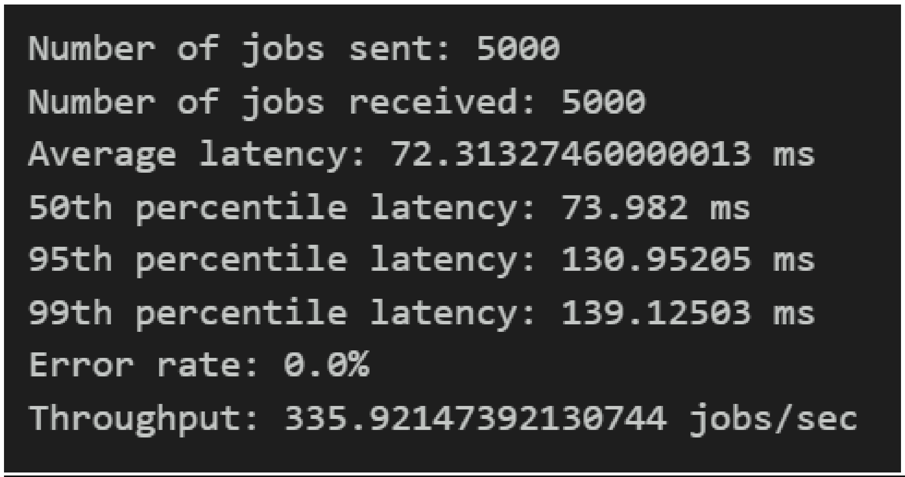
2. Research Flask integration (01/28/2024)

Providing insights into the operational aspects of Flask integration and guiding the implementation of this information within our project. This includes delivering instructions on the integration of databases, third-party libraries, and the incorporation of diverse technologies.

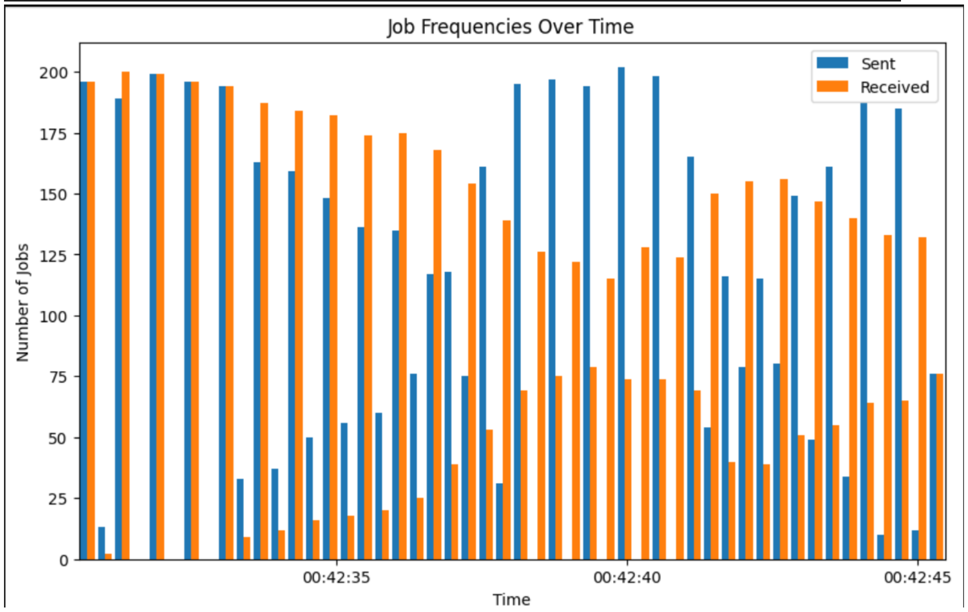
Done:

3. QA testing of parser module

Participating in an exhaustive and systematic Quality Assurance (QA) testing process for the parser module requires placing substantial emphasis on a thorough evaluation of its reliability, functionality, and performance. This meticulous testing approach involves a detailed examination of the module's codebase, ensuring compliance with predefined standards, scrutinizing responsiveness, and assessing overall operational efficiency.

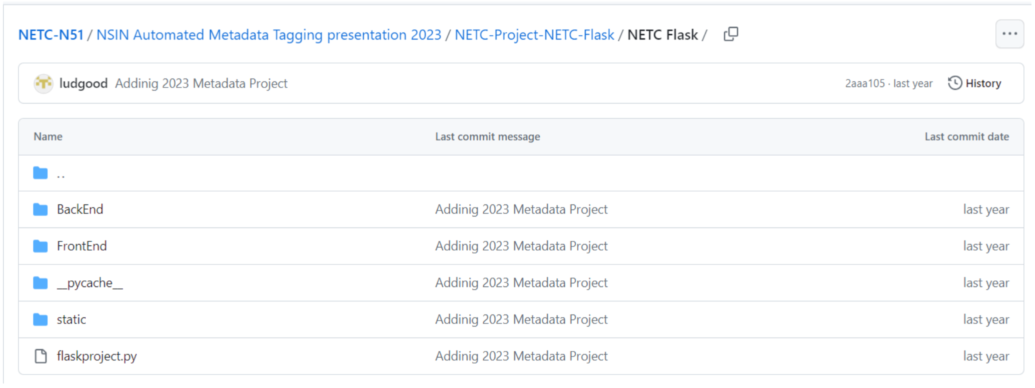


```
Number of jobs sent: 5000
Number of jobs received: 5000
Average latency: 72.31327460000013 ms
50th percentile latency: 73.982 ms
95th percentile latency: 130.95205 ms
99th percentile latency: 139.12503 ms
Error rate: 0.0%
Throughput: 335.92147392130744 jobs/sec
```



4. Assemble documentation of Flask development

Undertake the comprehensive endeavor of crafting detailed reports that meticulously document the nuances of the Flask development process. Focus on elucidating the origin and evolution of the Flask framework, delving into its fundamental concepts, and offering a thorough account of the creative and technical journey involved in bringing this Python web framework to fruition.



NETC-N51 / NSIN Automated Metadata Tagging presentation 2023 / NETC-Project-NETC-Flask / NETC Flask /

ludgood Addinig 2023 Metadata Project 2aaa105 · last year History

Name	Last commit message	Last commit date
..		
BackEnd	Addinig 2023 Metadata Project	last year
FrontEnd	Addinig 2023 Metadata Project	last year
__pycache__	Addinig 2023 Metadata Project	last year
static	Addinig 2023 Metadata Project	last year
flaskproject.py	Addinig 2023 Metadata Project	last year

In-Progress:

5. Assemble completed QA report for sponsor meeting

Compile and present the finalized Quality Assurance (QA) report in preparation for the upcoming sponsor meeting. Ensure that the report encompasses a comprehensive overview of the testing process, detailing key findings, identified issues, implemented fixes, and any relevant recommendations. Also, include clear and concise summaries of the testing outcomes, emphasizing the project's overall quality and reliability.

6. Research data streaming for abstraction layer.

Conduct thorough research on data streaming techniques and technologies tailored specifically for the abstraction layer. Explore and analyze various data streaming frameworks, protocols, and best practices to identify optimal solutions that align with the project's requirements.

Name of the Member: Team Mahmood Sakib
Report Date: 1/28/2024

Previous:

1. Develop QA testing for parser module (01/25/2024)

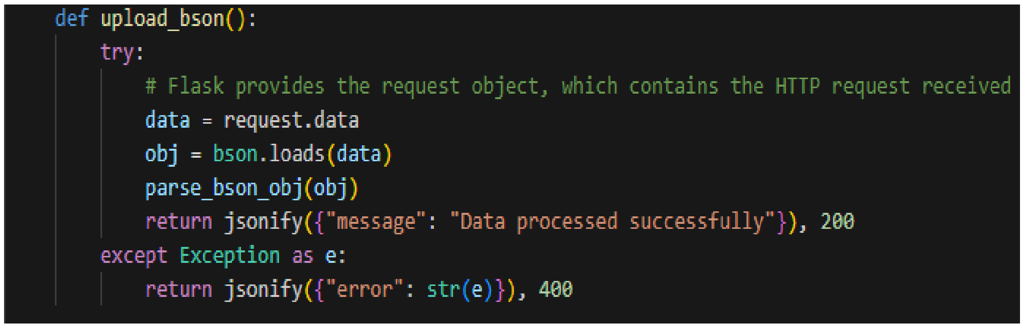
Conduct research and practical testing to formulate QA testing plans tailored for the parser module, placing specific emphasis on evaluating its reliability, functionality, and performance.

2. Continue parser module development (01/28/2024)

Advance the development of the parser module by incorporating insights obtained through research and testing to enhance its features. Focus on improving the module's functionality, ensuring reliability, and optimizing performance to create a resilient and efficient component.

Done:

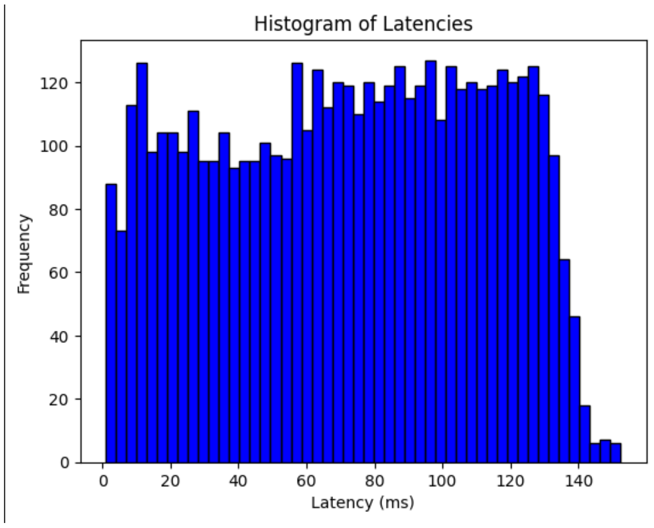
3. Assist in Flask integration



```
def upload_bson():
    try:
        # Flask provides the request object, which contains the HTTP request received
        data = request.data
        obj = bson.loads(data)
        parse_bson_obj(obj)
        return jsonify({"message": "Data processed successfully"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400
```

Participating in the integration process entails actively facilitating the assimilation of the Flask framework into the software, leveraging the capabilities of this Python-based web framework. This collaborative endeavor seeks to streamline and elevate web development, introducing simplicity and scalability to the overall process.

4. QA testing of the parser module



Executing a thorough Quality Assurance (QA) testing procedure for the parser module involves a detailed examination focused on assessing and confirming its reliability, functionality, and performance. This rigorous testing process includes scrutinizing the module's codebase, systematically evaluating its responsiveness and accuracy, and thoroughly examining its overall operational efficiency. The goal is to ensure a high standard of quality and effectiveness in the functionality of the module.

In-Progress:

5. Start proof-of-concept for data stream of message broker

Initiate the proof-of-concept phase for the data stream implementation of the message broker, encompassing the exploration of various technologies, frameworks, and methodologies. This stage involves the development of a prototype to validate the feasibility and effectiveness of integrating a data streaming mechanism into the existing message broker infrastructure.

6. Confirm fixes to the message broker based on QA report

Verify and implement the necessary fixes to the message broker in accordance with the findings and recommendations outlined in the Quality Assurance (QA) report. This process involves a comprehensive review of the reported issues, ensuring that each identified concern is addressed thoroughly.

Name of the Member: Aaron Steig
Report Date: 1/28/2024

Previous:

1. Compile weekly team reports (01/25/2024)

Create reports summarizing the discussions held during team-sponsor meetings, incorporating updates on the progress of the implementation.

2. Research Flask integration (01/28/2024)

Offering insights into the operational facets of Flask integration and instructing on the application of this knowledge within our project. This guidance encompasses instructions for integrating databases, incorporating third-party libraries, and adopting various technologies.

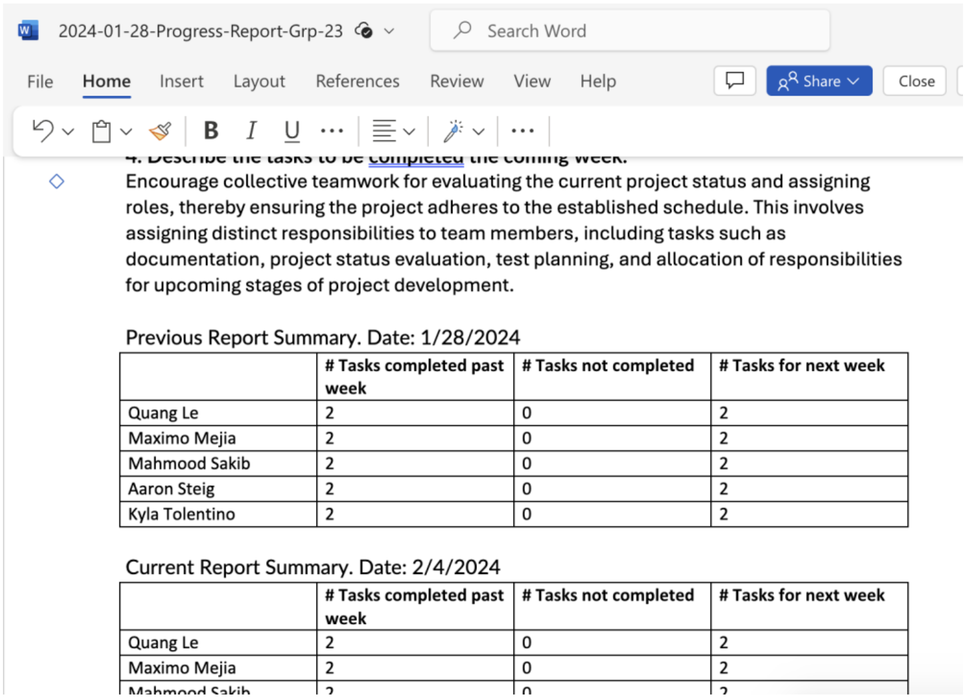
Done:

3. Initial implementation of the abstraction layer

Highlighting the conversion of abstract concepts into concrete code and structures, serving as a cohesive link among diverse components within the software system. Ensure that the design aligns with the overall system architecture and meets the project's requirements.

4. Complete team progress report

Establish a reiteration within the team to ensure that each team member consistently concludes the preparation of their weekly progress reports with meticulous attention, emphasizing the creation of comprehensive and accurate documentation.



4. Describe the tasks to be completed the coming week.

Encourage collective teamwork for evaluating the current project status and assigning roles, thereby ensuring the project adheres to the established schedule. This involves assigning distinct responsibilities to team members, including tasks such as documentation, project status evaluation, test planning, and allocation of responsibilities for upcoming stages of project development.

Previous Report Summary. Date: 1/28/2024

	# Tasks completed past week	# Tasks not completed	# Tasks for next week
Quang Le	2	0	2
Maximo Mejia	2	0	2
Mahmood Sakib	2	0	2
Aaron Steig	2	0	2
Kyla Tolentino	2	0	2

Current Report Summary. Date: 2/4/2024

	# Tasks completed past week	# Tasks not completed	# Tasks for next week
Quang Le	2	0	2
Maximo Mejia	2	0	2
Mahmood Sakib	2	0	2

In-Progress:

5. Develop initial API for abstraction layer

Embark on the initial stages of developing the API for the abstraction layer by defining key functionalities, designing essential endpoints, and outlining the fundamental structure. This process involves careful consideration of the layer's requirements, integration points, and scalability.

6. Research data stream for abstraction layer

Engage in a thorough investigation of data streaming technologies, specifically concentrating on their relevance to the abstraction layer. This research entails examining different data streaming frameworks, protocols, and industry-standard practices.

Name of the Member: Kyla Tolentino
Report Date: 1/28/2024

Previous:

1. Assist with parser module development (01/25/2024)

Advance the development of the parser module by incorporating insights derived from research and testing to elevate its features. Emphasize the refinement of the module's functionality, ensuring reliability and optimizing performance to establish a resilient and efficient component.

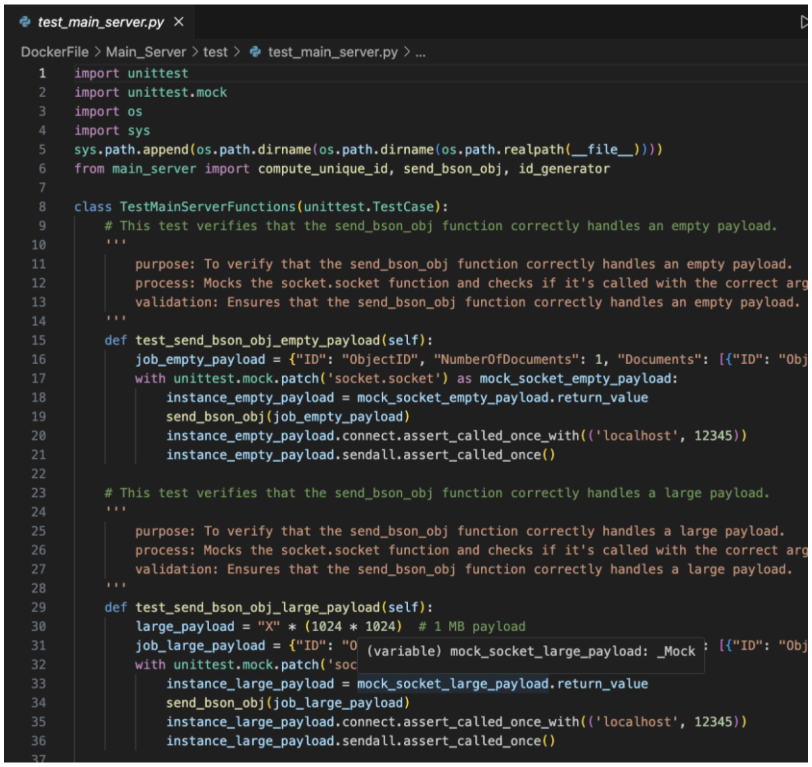
2. Ensure completion of progress reports (01/28/2024)

Guarantee that team members complete comprehensive and accurate documentation in their weekly progress reports, incorporating any final adjustments, well before the deadline.

Done:

3. Confirm adjustments on message broker based on QA test results

Verify and execute essential adjustments to the message broker system based on the insights and outcomes derived from the Quality Assurance (QA) testing process. This iterative refinement procedure entails a comprehensive review of QA test results, identification of areas for improvement or correction, and subsequent validation and application of necessary modifications to enhance the overall performance and reliability of the message broker.



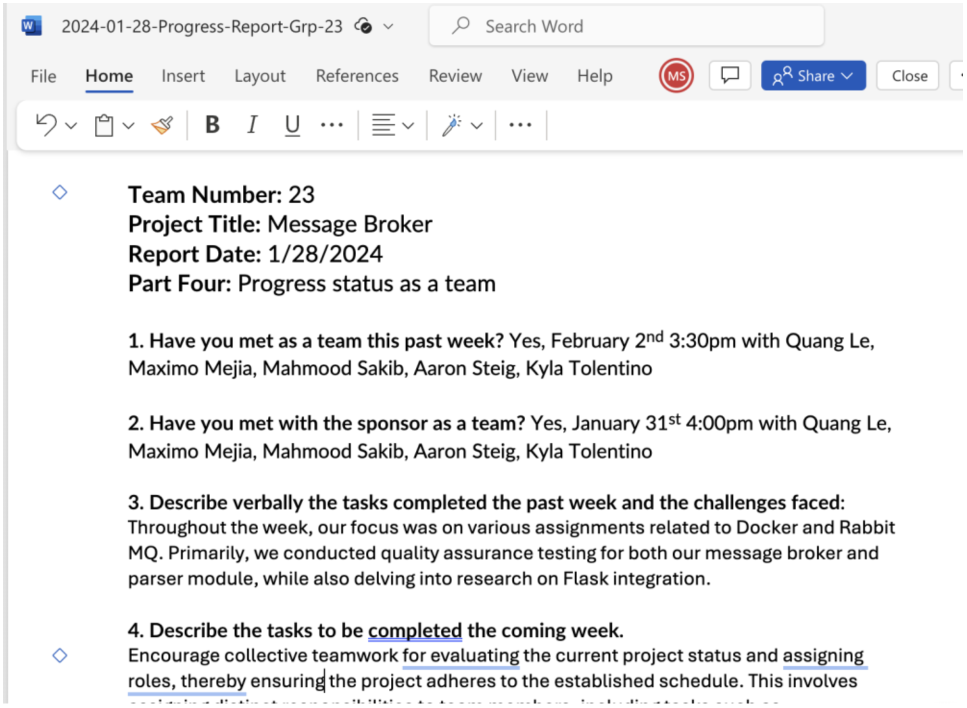
test_main_server.py

DockerFile > Main_Server > test > test_main_server.py > ...

```python
import unittest
import unittest.mock
import os
import sys
sys.path.append(os.path.dirname(os.path.dirname(os.path.realpath(__file__))))
from main_server import compute_unique_id, send_bson_obj, id_generator

class TestMainServerFunctions(unittest.TestCase):
    # This test verifies that the send_bson_obj function correctly handles an empty payload.
    '''
        purpose: To verify that the send_bson_obj function correctly handles an empty payload.
        process: Mocks the socket.socket function and checks if it's called with the correct arg
        validation: Ensures that the send_bson_obj function correctly handles an empty payload.
    '''
    def test_send_bson_obj_empty_payload(self):
        job_empty_payload = {"ID": "ObjectID", "NumberOfDocuments": 1, "Documents": [{"ID": "Obj
        with unittest.mock.patch('socket.socket') as mock_socket_empty_payload:
            instance_empty_payload = mock_socket_empty_payload.return_value
            send_bson_obj(job_empty_payload)
            instance_empty_payload.connect.assert_called_once_with(('localhost', 12345))
            instance_empty_payload.sendall.assert_called_once()

    # This test verifies that the send_bson_obj function correctly handles a large payload.
    '''
        purpose: To verify that the send_bson_obj function correctly handles a large payload.
        process: Mocks the socket.socket function and checks if it's called with the correct arg
        validation: Ensures that the send_bson_obj function correctly handles a large payload.
    '''
    def test_send_bson_obj_large_payload(self):
        large_payload = "X" * (1024 * 1024)  # 1 MB payload
        job_large_payload = {"ID": "O (variable) mock_socket_large_payload: _Mock : [{"ID": "Obj
        with unittest.mock.patch('soc
            instance_large_payload = mock_socket_large_payload.return_value
            send_bson_obj(job_large_payload)
            instance_large_payload.connect.assert_called_once_with(('localhost', 12345))
            instance_large_payload.sendall.assert_called_once()
```

4. Complete team progress report

Instill a commitment within the team to ensure the regular submission of weekly progress reports by every team member. Stress the significance of paying meticulous attention to detail, especially in crafting comprehensive and precise documen



Team Number: 23
Project Title: Message Broker
Report Date: 1/28/2024
Part Four: Progress status as a team

1. Have you met as a team this past week? Yes, February 2nd 3:30pm with Quang Le, Maximo Mejia, Mahmood Sakib, Aaron Steig, Kyla Tolentino

2. Have you met with the sponsor as a team? Yes, January 31st 4:00pm with Quang Le, Maximo Mejia, Mahmood Sakib, Aaron Steig, Kyla Tolentino

3. Describe verbally the tasks completed the past week and the challenges faced:
Throughout the week, our focus was on various assignments related to Docker and Rabbit MQ. Primarily, we conducted quality assurance testing for both our message broker and parser module, while also delving into research on Flask integration.

4. Describe the tasks to be completed the coming week.
Encourage collective teamwork for evaluating the current project status and assigning roles, thereby ensuring the project adheres to the established schedule. This involves

In-Progress:

5. Develop initial API for abstraction layer

Initiate the creation of the initial API for the abstraction layer by outlining core functionalities, defining essential endpoints, and establishing a foundational framework. The preliminary API development sets the groundwork for seamless communication among various components, with a focus on building a resilient and flexible interface aligned with the overarching project goals.

6. Start proof-of-concept for data stream in abstraction layer

To complete the exploratory phase for implementing a data stream in the message broker. This encompassed a comprehensive investigation into various technologies, frameworks, and methodologies. The initial proof-of-concept serves as a foundational step to gather insights, identify challenges, and inform strategic decisions for the successful integration of a data stream within the abstraction layer.