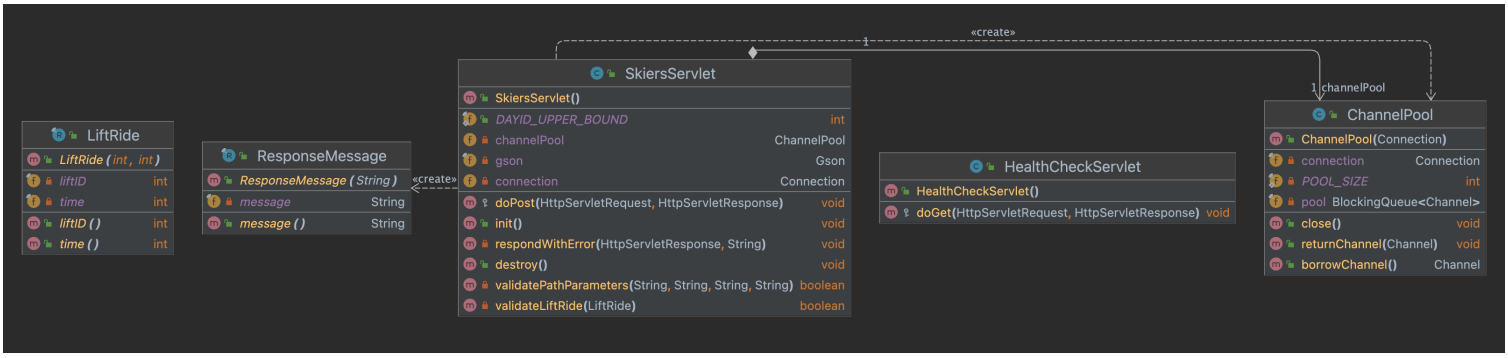
Assignment 2 Report

git-repo-URL:

https://github.com/yty20/CS6650-project

description:



Server Design Overview

The server architecture is constructed within the org.CS6650 package, which encapsulates components for managing HTTP requests, processing lift ride data, and interacting with a RabbitMQ message queue. The design follows a servlet-based model, commonly used in Java EE environments, to handle web requests and facilitate message queuing operations.

Core Components

- ChannelPool Class: This class is responsible for managing a pool of channels to RabbitMQ, allowing for reuse and efficient management of resources. It pre-creates a fixed number of channels (POOL_SIZE) and offers functionalities to borrow and return channels. It ensures that all channels are properly closed upon server shutdown.
- HealthCheckServlet Class: This servlet is a simple health check endpoint, returning an "OK" response. It is mapped to the URL pattern /api/* and serves as a diagnostic tool to quickly ascertain if the server is running.
- LiftRide Record: Defined as a record in Java, LiftRide encapsulates data related to a skier's ride on a lift, containing attributes for time and lift ID.
  ResponseMessage Record: Similarly, this record is used to structure JSON responses with a

message field, adhering to a consistent response format for the client.

- SkiersServlet Class: The primary servlet that handles POST requests on the URL pattern /skiers/*. It validates path parameters and the body of the request, constructs a message, and then publishes it to a RabbitMQ queue named "my_queue". This servlet also manages the lifecycle of the ChannelPool and the RabbitMQ connection.

Interactions and Data Flow

- HTTP Requests: Clients send requests to the servlets. HealthCheckServlet responds to GET requests for server health status, whereas SkiersServlet accepts POST requests with lift ride data.
- Validation: The SkiersServlet performs thorough validation on path parameters and JSON body to ensure the integrity of the data being processed.
- Message Queuing: Once validated, the lift ride data is serialized into a message string, which is then published to a RabbitMQ queue via a channel obtained from ChannelPool. This decouples the HTTP request processing from the message handling, allowing for asynchronous processing and potentially better scalability.
- Error Handling: In case of invalid data or server issues, error messages are generated using the ResponseMessage record and sent back to the client with an appropriate HTTP status code.
- Resource Management: The ChannelPool provides a robust mechanism for managing RabbitMQ channels, which are scarce resources. It encapsulates the complexity of pooling with thread-safe operations.
- Server Lifecycle: The SkiersServlet overrides the init and destroy methods to set up and tear down the connection to RabbitMQ, respectively, ensuring that resources are appropriately allocated and released.

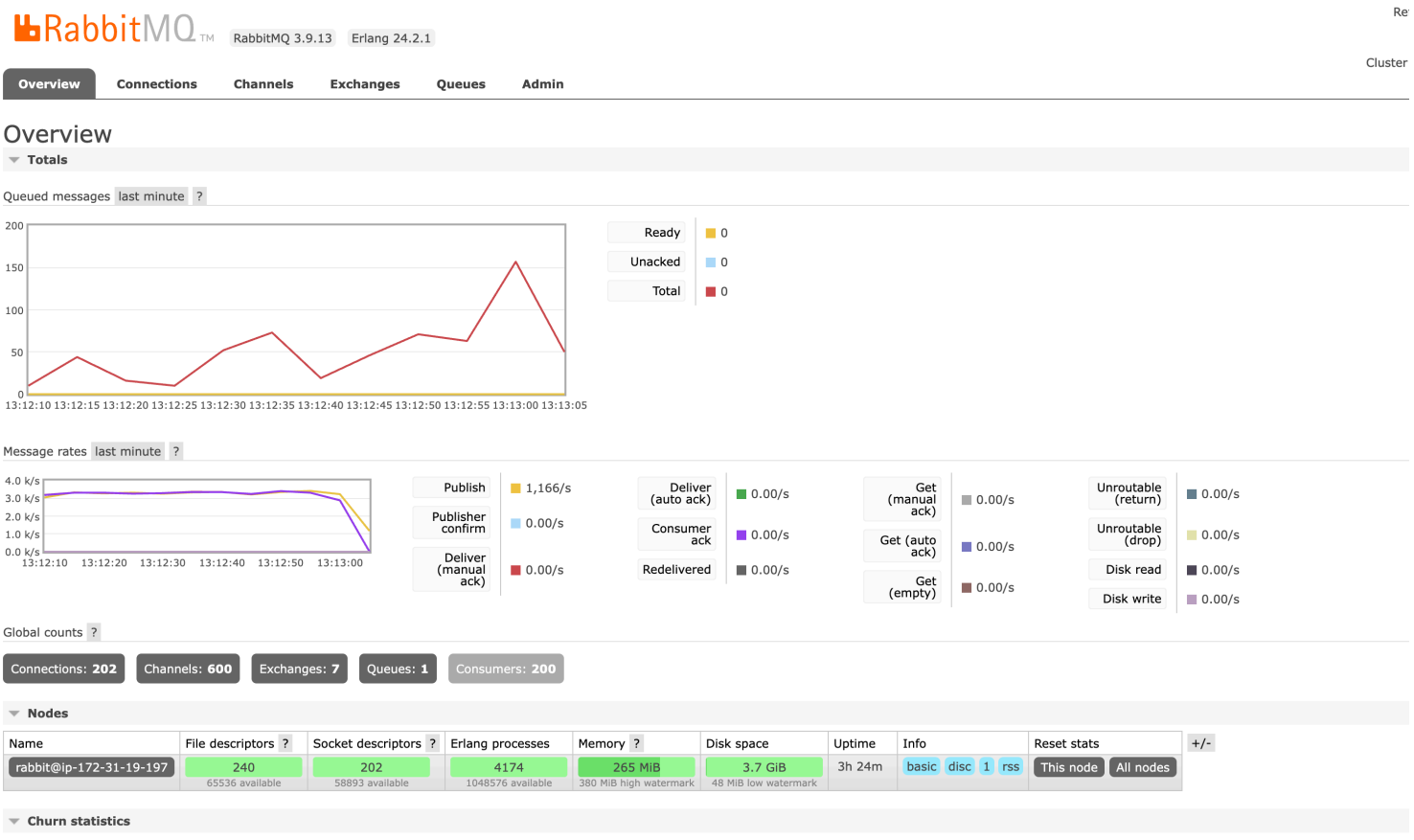
Package Structure

All classes are encapsulated within the org.CS6650 package, promoting modularity and namespace management. This structure makes it easy to locate service-related classes and can be beneficial for maintenance and scalability.

Results:

results with load balancer:

```
Total Successful Requests: 200000
Total Failed Requests: 0
Total Time: 64.214 s
Total throughput: 3114.5856043853364 requests/s
Latency Statistics:
Mean Latency: 63.907275 ms
Median Latency: 59 ms
Min Latency: 36 ms
Max Latency: 1581 ms
99th Percentile Latency: 126 ms
```



results without load balancer:

```
Total Successful Requests: 200000
Total Failed Requests: 0
Total Time: 88.041 s
Total throughput: 2271.668881543826 requests/s
Latency Statistics:
Mean Latency: 87.55547 ms
Median Latency: 66 ms
Min Latency: 35 ms
Max Latency: 1273 ms
99th Percentile Latency: 378 ms
```