**Server Design Description**

The server architecture consists of three main components:

1. Servlet Application: Handles incoming HTTP requests from clients, validates them, and publishes messages to a RabbitMQ queue.
2. Message Broker (RabbitMQ): Acts as a mediator for sending messages between the producer (servlet) and consumers.
3. Consumer Application: Multithreaded Java application that consumes messages from the queue and processes lift ride data.

These components are deployed on separate EC2 instances to distribute the workload and enhance scalability.

**Packages and Major Classes**

**Servlet Application**

**Package**: org.mliu

**SkierServlet**: This is the main servlet class that handles incoming POST requests. Extends HttpServlet and is annotated to handle /skiers/\* endpoints. It includes two methods.

init(): Sets up the RabbitMQ connection factory, establishing configurations like host, port, and credentials, and ensuring RabbitMQ connection availability across requests.

doPost(HttpServletRequest request, HttpServletResponse response): Handles incoming POST requests with the following steps:

* + - * 1. URL and Payload Validation: Verifies that path parameters (e.g., resortID, seasonID) and JSON payload fields (time, liftID) meet the expected format and values.
        2. Payload Parsing: Uses ObjectMapper (from Jackson) to deserialize the JSON payload into a LiftRide object.
        3. Message Serialization and Queueing: It serializes the LiftRide object to JSON format and publishes the message to the RabbitMQ queue using the established channel.

**LiftRide**: A model class representing the lift ride data which has fields of  resortID, seasonID, dayID, skierID, time, liftID. This method getters and setters for each field.

**Consumer Application**

**LiftRideConsumer**: Main class that initializes consumer threads to process messages from the queue.

It has two major fields, static int THREAD\_POOL\_SIZE defines the number of threads for concurrent message processing.

ConcurrentHashMap<Integer, CopyOnWriteArrayList<LiftRide>> skierLiftRides stores lift rides per skier ID, allowing concurrent access and modification.

**Methods**:

main(String[] args): Initializes the RabbitMQ connection and channel, then sets up an ExecutorService with a fixed thread pool based on THREAD\_POOL\_SIZE.

processMessage(String message): Deserializes the JSON message back into a LiftRide object and stores it in skierLiftRides. Uses computeIfAbsent() to initialize a list for each new skierID and adds lift ride data.

setupConsumerThreads(): Creates a consumer for each thread, binds each to the "LiftRideQueue", and handles message acknowledgment..

**Flow of Data and Message Processing**

1. Client Request: A client sends a POST request to the servlet with lift ride details (e.g., skierID, resortID, liftID, time).
2. Servlet Validation and Message Publishing: The servlet (SkierServlet) validates the request, parses the payload, and creates a LiftRide object. The LiftRide object is serialized to JSON and published to the "LiftRideQueue" queue in RabbitMQ.
3. Message Queueing: RabbitMQ receives and queues the message. Messages remain in the queue until a consumer is ready to process them, decoupling the servlet from the consumer application.
4. Consumer Message Processing: Each thread in the consumer application (LiftRideConsumer) retrieves messages from RabbitMQ.

processMessage(String message): Deserializes each JSON message into a LiftRide object.

Data Storage: The LiftRide is stored in a ConcurrentHashMap with skierID as the key, allowing efficient, concurrent access by multiple threads.

1. Message Acknowledgment: After processing, the consumer thread sends an acknowledgment back to RabbitMQ, which removes the message from the queue.

**Test run results without load balancer:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Test run results with load balancer starting at 15:45:  
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated